VŠB – Technická univerzita Ostrava
Fakulta strojní
Katedra automatizační techniky a řízení

Využití knihovny OpenCV v jazyce Python

Using the OpenCV library in Python

Student: Ladislav Rapan
Vedoucí práce: doc. Ing. Marek Babiuch, Ph.D.

Ostrava 2020

VŠB - Technická univerzita Ostrava
Fakulta strojní
Katedra automatizační techniky a řízení

Zadání bakalářské práce

Student: **Ladislav Rapan**

Studijní program: B2341 Strojírenství

Studijní obor: 3902R001 Aplikovaná informatika a řízení

Téma: Využití knihovny OpenCV v jazyce Python
Using the OpenCV Library in Python

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Seznamte se s jazykem Python pro platformy PC a Raspberry Pi.
2. Popište možnosti využití knihovny OpenCV.
3. Seznamte se s problematikou strojového učení za podpory knihovny OpenCV a dalších dostupných knihoven.
4. Vytvořte demonstrační úlohy rozpoznávání obrazu s použitím kamery s dostupnými knihovnami pro platformy PC a Raspberry Pi.
5. Zhodnoťte dosažené výsledky a navrhněte směry dalšího řešení.

Seznam doporučené odborné literatury:

UPTON, Eben a Gareth HALFACREE, 2013. *Raspberry Pi: uživatelská příručka.* Brno: Computer Press. ISBN 978-80-251-4116-8.
BRADSKI, Gary R. a Adrian KAEHLER, c2008. *Learning OpenCV.* Sebastopol: O'Reilly. ISBN 978-0-596-51613-0.
SUMMERFIELD, Mark, c2010. *Programming in Python 3: a complete introduction to the Python language.* 2nd ed., Fully rev. ed. Upper Saddle River, NJ: Addison-Wesley, Developer's library. ISBN 978-0-321-68056-3.
ALCHIN, Marty, c2010. *Pro Python.* New York: Distributed to the book trade worldwide by Springer Science+Business Media. Expert's voice in open source. ISBN 978-1-4302-2757-1.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Marek Babiuch, Ph.D.**

Datum zadání: 20.12.2019
Datum odevzdání: 18.05.2020

doc. Ing. Renata Wagnerová, Ph.D.
vedoucí katedry

VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA Fakulta strojní OSTRAVA

prof. Ing. Ivo Hlavatý, Ph.D.
děkan fakulty

Místopřísežné prohlášení

Prohlašuji, že jsem celou bakalářskou práci „Využití knihovny OpenCV v jazyce Python“ včetně příloh vypracoval samostatně pod vedením vedoucího bakalářské práce a uvedl jsem všechny použité podklady a literaturu.

V Bruntále dne 18. května 2020

Podpis autora práce

Prohlašuji, že:

- jsem si vědom, že na tuto moji závěrečnou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. Zákon o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (dále jen Autorský zákon), zejména § 35 (Užití díla v rámci občanských či náboženských obřadů nebo v rámci úředních akcí pořádaných orgány veřejné správy, v rámci školních představení a užití díla školního) a § 60 (Školní dílo),
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen „VŠB-TUO“) má právo užít tuto závěrečnou bakalářskou práci nekomerčně ke své vnitřní potřebě (§ 35 odst. 3 Autorského zákona),
- bude-li požadováno, jeden výtisk této bakalářské práce bude uložen u vedoucího práce,
- s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 Autorského zákona,
- užít toto své dílo, nebo poskytnout licenci k jejímu využití, mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše),
- beru na vědomí, že – podle zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů – že tato bakalářská práce bude před obhajobou zveřejněna na pracovišti vedoucího práce, a v elektronické podobě uložena a po obhajobě zveřejněna v Ústřední knihovně VŠB-TUO, a to bez ohledu na výsledek její obhajoby.

V Bruntále dne 18. května 2020

Rapan

Podpis autora práce

Jméno a příjmení autora práce: Ladislav Rapan

Adresa trvalého pobytu autora práce: Dělnická 15
Bruntál, 792 01

Poděkování

Tímto bych chtěl poděkovat panu doc. Ing. Marku Babiuchovi, Ph.D. za jeho ochotu při vedení této bakalářské práce, za to že mi umožnil věnovat se tomuto tématu a za cenné rady týkající se směru zpracování této práce.

Anotace bakalářské práce

RAPAN, L. *Využití knihovny OpenCV v jazyce Python: bakalářská práce*. Ostrava: VŠB – Technická univerzita Ostrava, Fakulta strojní, Katedra automatizační techniky a řízení, 2020, 45s. Vedoucí práce: doc. Ing. Marek Babiuch, Ph.D.

Tato práce se zaměřuje na práci s knihovnou OpenCV za použití jazyka Python. Jsou v ní znázorněny především možnosti této knihovny pro vizualizaci výsledků detekcí neuronových sítí. Práce s touto knihovnou je také ukázána na platformě Raspberry Pi. V poslední řadě jsou zde také naznačeny základní pojmy neuronových sítí. Jako příklady použití tato práce ukazuje využití knihovny pro detekci objektů na feedu z webkamery, za pomoci YOLO neuronové sítě a detekci obličejů za pomocí knihovny facenet-pytorch.

Klíčová slova: facenet-pytorch, neuronové sítě, OpenCV, Raspberry Pi, YOLO

Annotation of bachelor thesis

RAPAN, L. *Using the OpenCV library in Python: Bachelor Thesis*. Ostrava: VŠB – Technical University of Ostrava, Faculty of Mechanical Engineering, Department of Control Systems and Instrumentation, 2020, 45p. Thesis head: doc. Ing. Marek Babiuch, Ph.D.

This work deals with the usage of OpenCV library with Python. It shows capabilities of this library for visualization of neural network outputs. It also shows the possibility of using this library on the Raspberry Pi platform. The last part of this work is dedicated to basic concepts of neural networks. This work also shows example of usage for object detection on webcam feed using the YOLO network and facial detection using the facenet-pytorch library.

Key words: facenet-pytorch, neural networks, OpenCV, Raspberry Pi, YOLO

Obsah

Seznam použitých zkratek

BLOB – Binary Large Object

CNN – Convolutional neural network

COCO – Common Objects in Context

CPU – Central Processing Unit

CSI – Camera Serial Interface

CUDA – Compute Unified Device Architecture

DNN – Deep Neural Network

DSI – Display Serial Interface

GB – Gigabyte

GHz – Gigahertz

HDMI – High-Definition Multimedia Interface

IDE – Interactive Development Environment

IDLE – Integrated Development and Learning Environment

IT – Informační technologie

MTCNN – Multi-task Cascaded Convolutional Networks

OpenCV – Open Source Computer Vision Library

OS – Operační systém

PC – Personal Computer

PIL – Python Imaging Library

RAM – Random Access Memory

R-CNN – Regional Convolutional Neural Network

RPi – Raspberry Pi

SD – Secure Digital

SQL – Structured Query Language

tzv. – takzvaný

USB – Universal Serial Bus

YOLO – You Only Look Once

Úvod

Cílem této práce je do jisté míry ukázat možnosti využití knihovny OpenCV v kombinaci s programovacím jazykem Python a zprovoznění úlohy využívající tuto knihovnu na platformě Raspberry Pi.

První kapitola se věnuje základním informacím o jazyce Python a popisuje možnosti jeho použití v kombinaci s některými knihovnami. Následně obsahuje informace o použití tohoto programovacího jazyka při práci na různých operačních systémech a základní importování knihoven. Věnuje se také platformě Raspberry Pi. Jsou zde zmíněny hardwarové specifikace modelů, které použijeme v praktickém příkladu, a také základní informace o operačním systému této platformy.

Druhá kapitola je věnována samotné knihovně OpenCV. V úvodu kapitoly jsou zmíněny některé základní informace o této knihovně a její instalaci. Následně lze v této kapitole najít popisy některých funkcí obsažených v této knihovně, které jsou později použity v praktickém příkladu. Část této kapitoly se také věnuje základům neuronových sítí a ukazuje příklad architektury neuronové sítě spadající do kategorie Deep Learning. Poslední část této kapitoly je věnována praktickým příkladům. Jako první příklad je zde ukázáno využití knihovny pro detekci objektů na fotografii, pořízené pomocí web kamery, za použití YOLO neuronové sítě. Dalším příkladem je detekce objektů na videopřenosu z webkamery. Jako poslední část potom porovnáme čas detekce při práci na platformách PC a Raspberry Pi.

Třetí kapitola je věnována praktickému příkladu pracujícímu s živým přenosem obrazu z webkamery. Jádrem tohoto příkladu je použití dostupného modelu neuronové sítě pro detekci a identifikaci obličejů k rozpoznávání námi určených obličejů nalezených na snímcích, které programu dodává webkamera. Ukazuje také vizualizaci výsledků do získaných snímků a následné zobrazení na monitor. Tato kapitola popisuje i knihovnu facenet-pytorch, která zajišťuje neuronové sítě pro detekci a identifikaci, a využití dostupného skriptu pro přetrénování dodaného modelu na požadovaný dataset. Tento příklad následně rozšíříme o zobrazování a získávání některých informací o identifikovaných subjektech.

1 Python

Python je programovací jazyk, který vytvořil Guido Van Rossum a poprvé byl vydán v roce 1991. Jedná se o interpretovaný, objektově orientovaný skriptovací programovací jazyk. Jazyk Python vyšel v několika verzích, z nichž nejnovější je Python 3, konkrétně pak verze Python 3.8. Momentálně je Python 3 jedinou oficiálně podporovanou verzí, jelikož verzi 2 skončila podpora na konci roku 2019. Na rozdíl od jiných jazyků, jako například C#, se s Pythonem dá pracovat na všech větších OS (Windows, iOS, Linux). Soubory Pythonu se ukládají jako soubory s koncovkou *.py*. (1)

Tento programovací jazyk najde uplatnění ve spoustě odvětví informačních technologií a je oblíbený jak u začínajících programátorů a IT nadšenců, tak u profesionálů, pracujících v různých specializovaných odvětvích. Mezi jeho výhody se nepochybně řadí jednoduchost jeho syntaxí a uživatelská přívětivost, syntaxe tohoto jazyka totiž nemají daleko k pseudokódu, a proto i začínající programátor z kódu dokáže vyčíst, co přesně program bude dělat. Dalším důvodem k použití tohoto jazyku je nespočet knihoven, které je možno po instalaci využít, ať už se jedná o knihovny používané k programování her, jako například knihovna PyGame, knihovny používané k matematickým výpočtům (knihovna NumPy), nebo knihovny používané k práci s obrazem, strojovému učení a neurálním sítím, jako knihovna OpenCV. (1)

1.1 Python na platformě Windows

Pro práci s jazykem Python je potřeba mít Python nainstalovaný. Python jako Open-source jazyk je volně dostupný na oficiální webové stránce www.python.org. Na této stránce lze najít většinu verzí pythonu, a to jak Pythonu 3, tak Pythonu 2.

V pythonu lze pracovat dvěma způsoby. Prvním způsobem je programování v Shell okně, jedná se o interaktivní okno, pracující podobně jako live skript v programu Matlab. Programátor zadá příkaz a počítač jej hned realizuje. Odpadá tak nutnost před každým spuštěním programem kompilovat program. Tento způsob je vhodný pro jednodušší programy, které nevyžadují složitý a dlouhý kód.

Druhým způsobem je programování ve vývojovém prostředí. Těchto prostředí existuje spousta a výběr závisí pouze na preferenci uživatele. Základním vývojovým prostředím je Python IDLE editoru, které obsahuje instalační balíček Pythonu. Při programování tímto způsobem uživatel napíše kód celého programu, následně jej zkompiluje a až po tom je program vykonán. Tento způsob je vhodný především pro složitější a delší kódy. Jako příklad dalších IDE prostředí používaných pro programování v Python bych chtěl dále

uvést IDE PyCharm od vývojového studia JetBrains, které je velmi podobné prostředí Python IDLE, umožňuje však lepší přizpůsobení podle chuti vývojáře.

Skripty napsané v IDLE a uložené je následně možné vyvolat a spustit z příkazového řádku, nebo je možné je spouštět přímo z prostředí IDLE.

1.2 Python v operačních systémech založených na platformě Linux

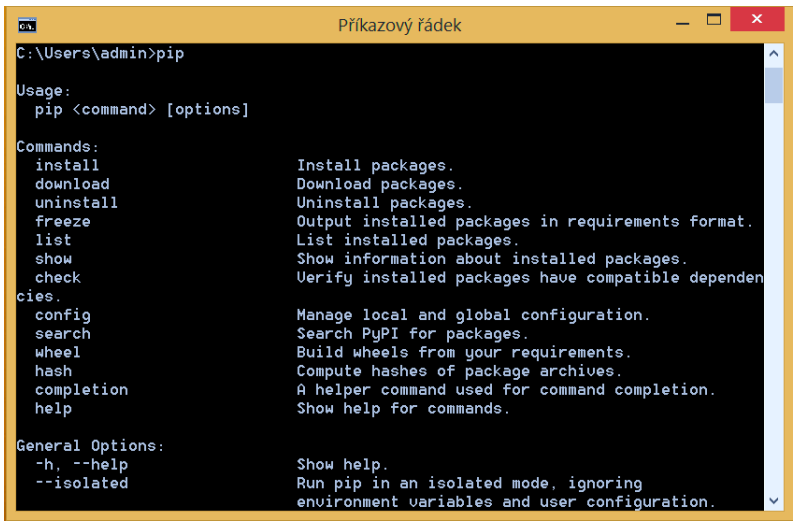
Instalace Pythonu v některých OS založených na platformě Linux není potřeba, Python v nich totiž bývá zakomponován od začátku. V případě, že na OS Python nainstalován není, bude se jeho instalace lišit podle typu operačního systému. Snadným způsobem instalace Pythonu je instalace skrze příkazový řádek, kde se instalace skládá z několika jednoduchých příkazů.

Použití samotného Pythonu se oproti jeho použití v systémech Windows moc neliší. Rozdíly se objeví až při práci s knihovnami, některé knihovny jsou totiž vytvořeny pro práci ve specifických systémech a je proto potřeba hledat alternativní knihovnu, která funguje i v požadovaném systému.

1.3 Práce s knihovnami v jazyce Python

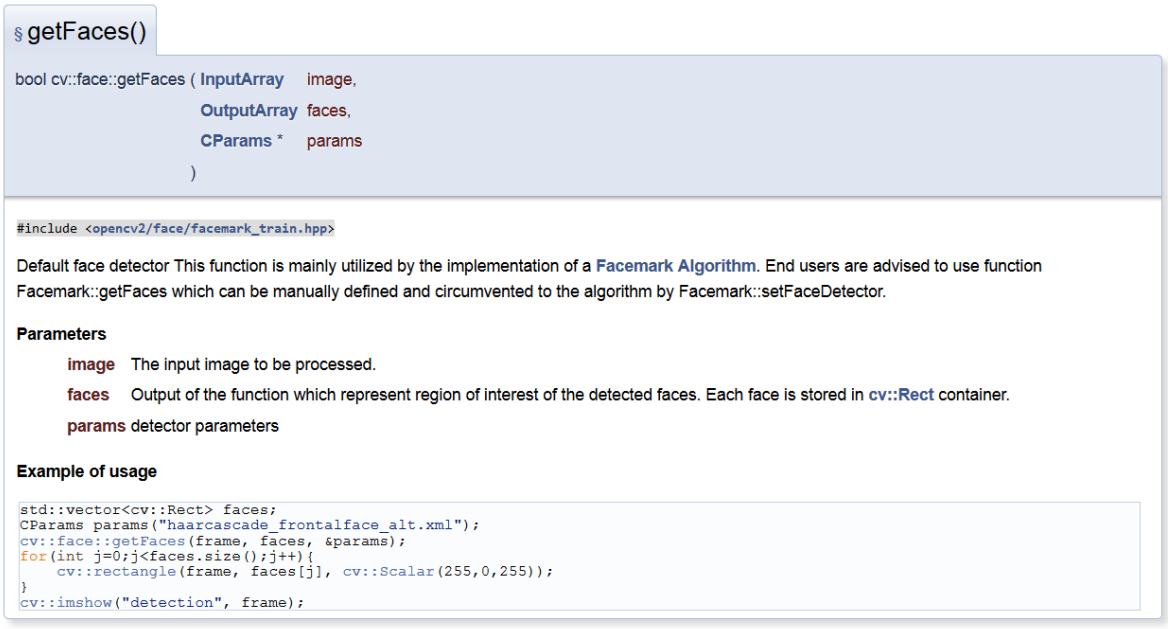
Jako většina programovacích jazyků i Python umožňuje pracovat s knihovnami, které usnadňují práci programátorům. Knihovny jsou balíčky, které nám umožní ulehčit si práci využitím kódu, který už byl napsán. V praxi to znamená, že nemusíme psát funkce a algoritmy, které jsou v dané knihovně k dispozici a k jejich použití nám stačí importovat knihovnu do našeho kódu a následně funkci vyvolat, jako bychom volali jakoukoliv námi napsanou funkci.

Nejsnadnější, a zároveň doporučovanou, cestou instalace knihoven do Pythonu je použití nástroje *pip*. Jedná se o nástroj spouštěný z příkazového řádku, který stáhne a nainstaluje požadovanou knihovnu sám. Tento nástroj v základu vyhledává a stahuje balíčky z oficiálního úložiště Python, umožňuje ale i instalaci balíčků z jiných webových úložišť. (2)



Obrázek 1 Nástroj pip v příkazovém řádku

Pro správné použití knihovny je třeba se seznámit s dokumentací, kterou autoři knihovny napsali pro budoucí uživatele. Tato dokumentace je většinou k dispozici na webech těchto knihoven a dá se najít jak pro nejnovější verzi knihovny, tak pro starší verze. Ne všechny projekty vždy využívají tu nejnovější verzi dané knihovny a může se stát, že při práci na starším projektu zjistíme, že využívá verzi knihovny, která není aktuální. V tomto případě se většinou spíše vyplatí šáhnout po starší dokumentaci než přepisovat celý projekt pro práci s novější knihovnou.



§ getFaces()

bool cv::face::getFaces (InputArray image,
OutputArray faces,
CParams * params
)

#include <opencv2/face/facemark_train.hpp>

Default face detector This function is mainly utilized by the implementation of a Facemark Algorithm. End users are advised to use function Facemark::getFaces which can be manually defined and circumvented to the algorithm by Facemark::setFaceDetector.

Parameters

image The input image to be processed.

faces Output of the function which represent region of interest of the detected faces. Each face is stored in cv::Rect container.

params detector parameters

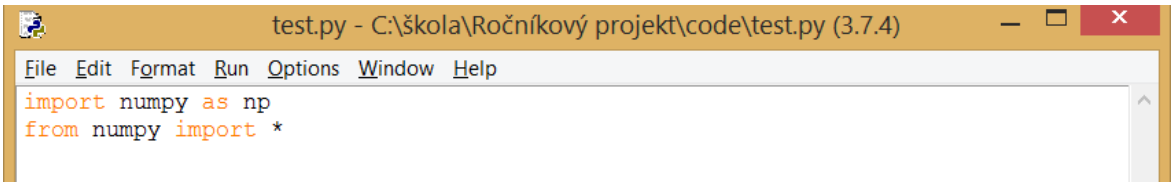
Example of usage

```
std::vector<cv::Rect> faces;
CParams params("haarcascade_frontalface_alt.xml");
cv::face::getFaces(frame, faces, &params);
for(int j=0;j<faces.size();j++){
    cv::rectangle(frame, faces[j], cv::Scalar(255,0,255));
}
cv::imshow("detection", frame);
```

Obrázek 2 Příklad dokumentace knihovny OpenCV (3)

Při programování se knihovny importují do kódu pomocí funkce import. Pokud je název funkce delší, můžeme si zjednodušit její následné volání pomocí příkazu *import „knihovna“ as „zkratka“*, viz příklad na obrázku. Toto nám umožní při následném volání knihovny

nepsat celý její název, ale pouze zkratku, pod kterou jsme ji importovali. Při volání funkcí z importované knihovny musíme před danou funkcí psát jméno knihovny nebo zkratku.



Obrázek 3 Python IDLE – importování knihoven

Další možností při importování knihovny je importování jen části větší knihovny. Tohoto docílíme příkazem *from „knihovna“ import „žádaná část knihovny“*. Při využití příkazu *from „knihovna“ import* * se importují všechny funkce z dané knihovny tak, že nebudeme muset při volání funkcí této knihovny první psát jméno této knihovny.

1.4 Raspberry Pi

Platforma Raspberry Pi je jednodeskový počítač, které díky své relativně nízké ceně a svým malým rozměrům najde široké uplatnění. Často bývá použit při domácích projektech, kdy jde použít jako domácí server, osobní počítač, řídicí jednotka embedded systémů nebo jako učební pomůcka. Uplatnění najde také jako pomůcka při penetračním testování sítí a systémů, kdy se jako jeho OS používá Kali Linux.

1.4.1 Raspberry Pi 3 Model B

V rámci této práce budeme pracovat s modelem Raspberry Pi 3 Model B. Tento model má k dispozici čtyřjádrový 64bitový procesor o frekvenci 1,2GHz, RAM pamětí 1 GB. Je vybaven Wi-Fi a Bluetooth modulem. CSI portem pro připojení kamery, DSI portem pro připojení toutchscreen displeje, Ethernet portem a HDMI portem pro připojení monitoru. K dispozici jsou také 4 USB porty pro připojení periférií nebo flash disku. Pro nahrání OS se používá slot pro microSD kartu. (4)

1.4.2 Raspberry Pi 4 Model B

Druhým modelem Raspberry Pi, kterým použijeme v této práci, je RPi 4 model B. Ten má k dispozici 4 jádrový 64bitový procesor o frekvenci 1,5 GHz, 4 GB RAM paměti, Bluetooth a Wi-Fi modul a také ethernetový port pro připojení k síti. Obsahuje také 2 USB 3.0 porty a 2 USB 2.0 porty, 2 micro-HDMI porty pro připojení 2 monitorů a CSI a DSI porty pro připojení kamery a displeje. OS načítáme z microSD karty. (5)

1.4.3 Operační systém a jeho instalace

Jak už bylo zmíněno, Raspberry Pi bootuje z microSD karty. Je tedy potřeba mít OS systém nachystaný na kartě. Obvykle se jako operační systém používá Raspbian, který je

volně dostupný z webových stránek produktu, dá se ale využít i jiných operačních systému jako například Kali nebo Ubuntu. Raspbian je operační systém zakládající se na sytému Debian. Debian je operační systém na bázi Linuxu.

Pro instalaci Raspbianu je potřeba stáhnout jednu z jeho verzí dostupným na webu Raspberry Pi. K dispozici jsou 3 obrazy tohoto systému a to Raspbian, Raspbian s doporučeným softwarem a Raspbian Lite. Abychom později nemuseli instalovat Python použijeme verzi s doporučeným softwarem. V následujícím kroku použijeme software balenaEtcher, který obraz OS zapíše na SD kartu. Následně kartu vložíme do slotu na Raspberry Pi a systém je připraven k použití. Jako po instalaci každého Linux systému je vhodně provést aktualizaci pomoci příkazů *sudo apt update a sudo apt upgrade*. (6)

2 Knihovna OpenCV

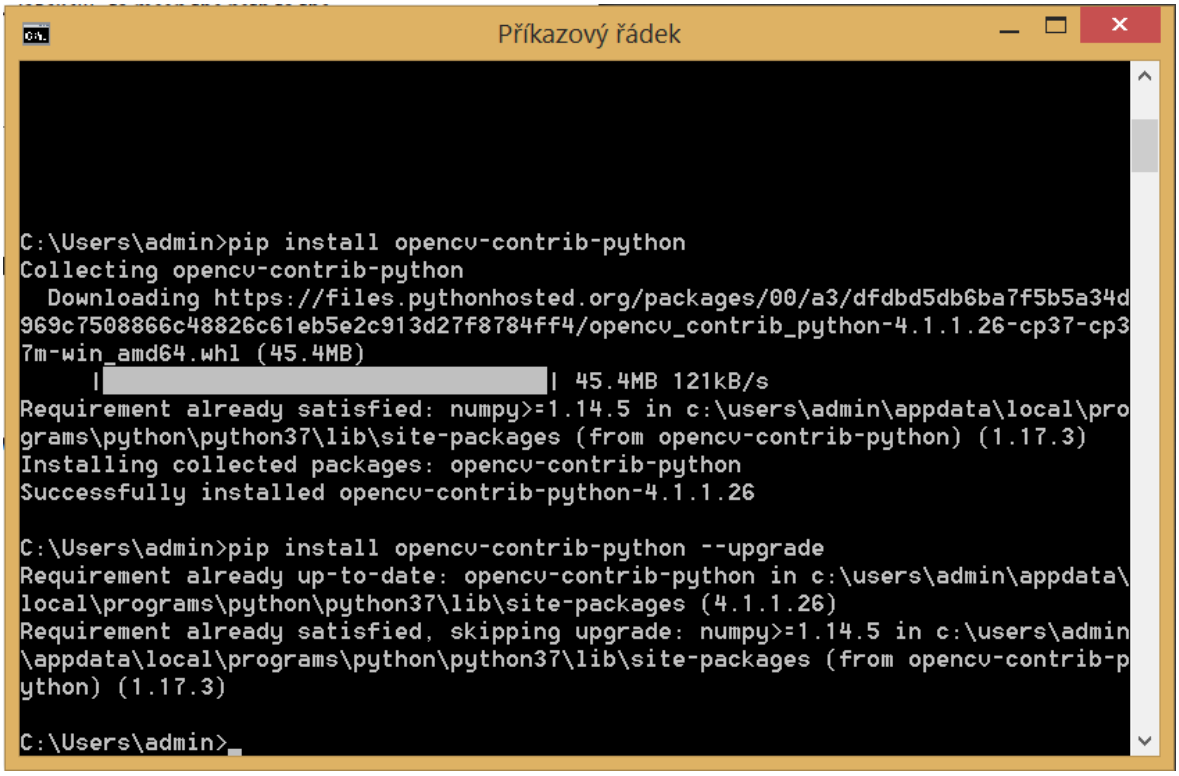
OpenCV – Open Source Computer Vision Library – je Open source knihovna zaměřená na strojové učení a počítačové vidění, obsahující přes 2500 algoritmů. Mezi její uživatele se řadí výzkumné týmy i firmy, které se touto problematikou zabývají. Knihovna OpenCV je původně psaná v jazyce C++, není ale limitovaná pouze pro použití v tomto jazyce, dá se použít i v jazycích Python, Java a MATLAB. Použití není limitováno operačními systémy, protože podporuje OS Windows, Linux, Android a Mac OS. Hlavním zaměřením je pak Real-time zpracování obrazu. Mezi největší firmy používající tuto knihovnu patří například Google, IBM, Yahoo, Microsoft nebo Honda. (7)

2.1 Instalace knihovny

Pro instalaci knihovny je možné použít instalační soubor dostupný na webových stránkách OpenCV, a to jak při instalaci na platformě Windows, tak na OS Raspbian. V případě problémů s touto instalací lze také knihovnu instalovat pomocí nástroje pip.

Při instalaci na OS Windows pomocí pip nástroje je postup následující. Jako první je v příkazovém řádku vhodné provést aktualizaci nástroje pip a jejich repositářů. Toho docílíme příkazem *python3 -m pip install --upgrade pip*. Následně nainstalujeme knihovnu OpenCV pro python a aktualizujeme ji příkazy *python3 -m pip install opencv-contrib-python* a *python3 -m pip install opencv-contrib-python --upgrade*. Po tomto by měla být knihovna připravena k použití. Budeme také potřebovat knihovnu numpy, kterou nainstaluje obdobým způsobem.

Při instalaci na Raspbian je postup víceméně stejný. Abychom příkazy zadávali jako root musíme před každý příkaz přidat slovo sudo. Příkaz pro aktualizaci pip tedy bude mít tvar *sudo python3 -m pip install --upgrade pip*.



Obrázek 4 Instalace OpenCV pomocí nástroje pip

V rámci této práce se budeme věnovat převážně dvěma možnostem využití této knihovny, a to práci s neuronovými sítěmi spadající do oblasti strojového učení Deep Learning a možnosti práce s obrazem. Pro strojové učení nabízí OpenCV moduly Machine Learning a DNN. Modul Machine Learning nabízí například funkce použitelné pro logické regrese a neuronové sítě nespadající do oblasti Deep Learning. Naproti tomu modul DNN nabízí možnost importování a použití neuronových sítí do této kategorie spadající, proto i jeho název DNN je zkratka z anglického označení těchto sítí Deep Neural Networks.

2.2 Zpracování obrazu

Použití OpenCV pro zpracování obrazu má poměrně široké množství možností. Od základního konvertování obrazu na pole pixelů, po zachycení obrazu z živého feedu webkamery a provádění různých operací na tomto zachyceném obraze. V této kapitole budou popsány funkce, které následně použijeme při tvorbě praktického příkladu.

imread("img.jpg")

Jak napovídá její název funkce *imread* slouží k převedení obrázku z formátů jako je například *jpg* nebo *png* do formátu použitelného pro další práci. Jako vstup do této funkce se zadává jméno souboru, ve kterém je obrázek uložen. Výstupem je matice, která vyjadřuje každý pixel, ze kterého se obrázek skládá jako kombinací 3 hodnot. Tyto hodnoty reprezentují barvy, ze kterého se pixel skládá. (8)

rectangle(img, (x_1, y_1),(x_2, y_2),(255,0,0), cv.FILLED)

Funkce *rectangle* slouží k vykreslení obdélníku do obrazu. Podobně jako funkce *rectangle* slouží k vykreslení různých tvarů několik dalších funkcí, které proto nemá cenu rozebírat. Jako vstup pro tuto funkci je potřeba zadat proměnnou obsahující konvertovaný obrázek, do kterého má být kresleno, pozici vrchního levého rohu v pixelech, pozici pravého dolního rohu v pixelech a barvu. V případě potřeby je možné měnit i šířku a typ čáry, kterou bude obdélník vykreslen. Pro vykreslení vyplněného obdélníku můžeme jako parametr thickness určující tloušťku obrysové čáry napsat cv.FILLED. Stojí za zmínku, že barva se musí zadávat v pořadí BGR, nikoliv RGB.

cv.putText(img, text, (x,y), fontface, fontscale, (255,0,0), thickness, lineType)

Funkce *putText* slouží k vykreslení textu do obrazu. Vstupní parametry této funkce:

- *img* – obraz, do kterého chceme vykreslovat.
- *text* – proměnná typu string, ve které je uložen vykreslovaný text.
- *(x,y)* – souřadnice levého spodního rohu obdélníku, ve kterém bude text umístěn.
- *fontface* – slouží k volbě fontu.
- *fontscale* – určuje velikost písma.
- *(B, G, R)* – barva textu.
- *thickness* – tloušťka textu.
- *lineType* – umožňuje měnit typ čáry, jakou bude text vykreslen (čárkovaná, čerchovaná…)

Fonty

Pro volání fontu je třeba použít jeden z fontů, které OpenCV podporuje, kompletní seznam je k dispozici v dokumentaci knihovny. Fonty jsou definovány pomocí klíčových slov jako například:

cv.FONT_HERSHEY_SIMPLEX

Volat tyto fonty pak jde jak pomocí těchto klíčových frází, tak pomocí čísel, která jsou těmto fontům přiřazená. Parametr *fontface* vyžaduje zápis fontu v proměnné int,

takže i v případě, že font voláme pomocí klíčových slov, jsou stejně převedeny do číselné podoby. Pro přehlednost je ale jednodušší pracovat s nimi pomocí jejich názvů. (9)

cv.imshow(img)

Funkce *imshow* slouží k zobrazení obrazu, uloženého v proměnné jako matici pixelů o určitých kombinacích barev, v klasickém grafickém formátu, srozumitelnému lidskému oku. Tato funkce ovšem slouží pouze k vykreslení obrázku, ne k jeho převedení a uložení do souboru. Je tedy vhodná zejména pokud není potřeba zpracovaný obraz ukládat pro další použití. (10)

2.3 OpenCV a Deep Learning

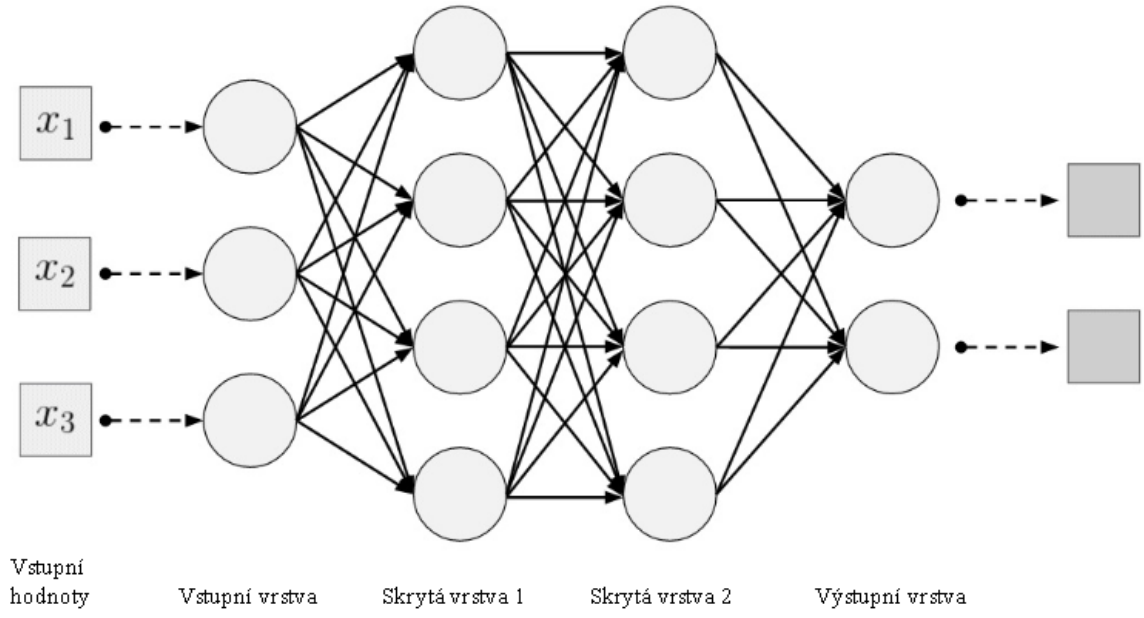
Jak už bylo dříve zmíněno podporu pro Deep Learning v OpenCV zajišťuje modul DNN. Tento modul obsahuje především funkce, které umožnují importování neuronových sítí připravených pomocí jiných nástrojů. Pro práci s ním je tedy potřeba mít k dispozici trénovanou neuronovou síť, která je charakterizována pomocí souboru obsahujícím informace o jednotlivých vrstvách sítě a souborem obsahující váhy jednotlivých synapsí.

2.3.1 Neuronové sítě

Neuronové sítě jsou počítačové modely, jejichž vnitřní struktura je podobná struktuře mozku. Mezi nejznámější použití těchto modelů pravděpodobně patří rozpoznávání obrazu. Vycvičené modely jsou schopny rozpoznávat ručně psaná čísla a písmena, objekty nacházející se na obraze, nebo obličeje. Využít je ale lze i v jiných oblastech, jako například v oblasti počítačové bezpečnosti nebo generování textu.

Architektura sítí

Architektura jednoduché sítě je zobrazena na obrázku 5. Neuronové sítě se skládají ze 3 základních typů vrstev. Jedná se o vstupní vrstvu, skryté vrstvy a výstupní vrstvu. Vstupní vrstva je vrstva, do které zadáváme vektor vstupních hodnot. Výstupní vrstva obsahuje vektor výstupních hodnot. To jsou hodnoty, které jsme získali použitím sítě na vstupní data. Každá síť může obsahovat libovolné množství skrytých vrstev, v těchto skrytých vrstvách probíhají výpočty, díky kterým získáme výstupní data. Jednotlivé vrstvy obsahují neurony, které jsou zobrazeny pomocí kruhů. Tyto neurony jsou spojeny s ostatními neurony pomocí synapsí.



Obrázek 5 Struktura neuronových sítí (11)

Neurony ve vstupní vrstvě jsou pomocí synapsí spojeny se všemi neurony, které se nachází v následující skryté vrstvě. Každý jednotlivý neuron v jakékoliv skryté vrstvě je spojen s každým neuronem jak v předcházející, tak následující vrstvě. Neurony ve výstupní vrstvě jsou spojeny s každým neuronem obsaženým v poslední skryté vrstvě. (11)

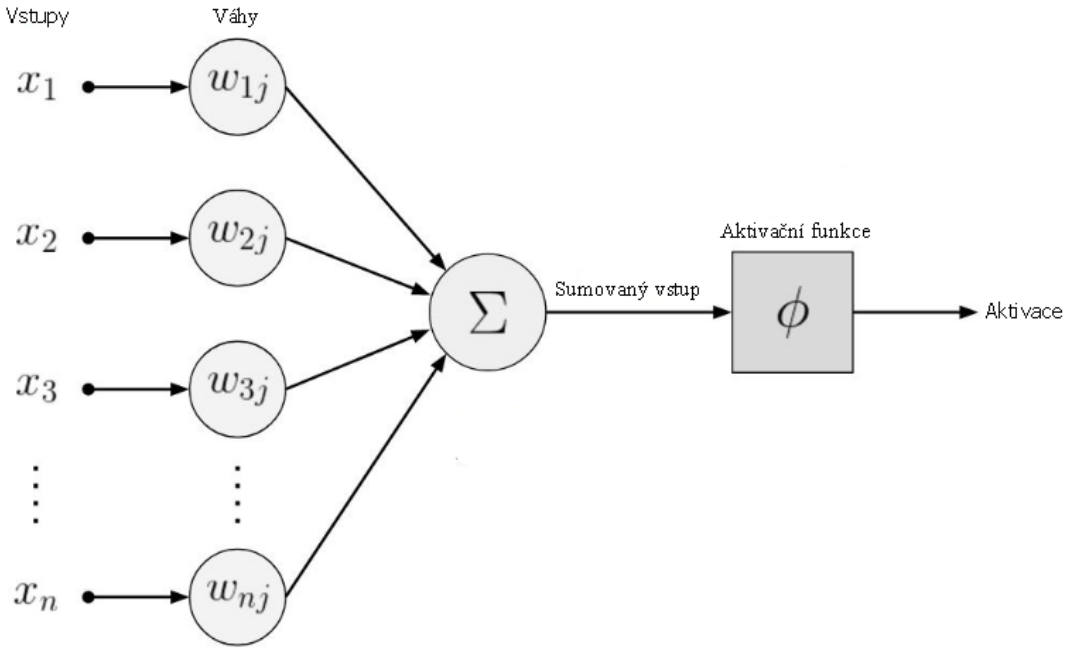
Synapse

Jak už bylo zmíněno, neurony jsou propojeny pomocí synapsí. Na obrázku jsou tyto synapse naznačeny pomocí šipek. Synapse v počítačovém modelu neuronových sítí hrají důležitou roli, jelikož slouží jako způsob, jakým síť rozhoduje, který neuron je důležitý pro získání správného výsledku a který ne. Prakticky jsou synapse reprezentovány pomocí vah. Váhy si síť upravuje při procesu cvičení. Pokud daný neuron pozitivně ovlivňuje výsledek, tak je váha s ním spojena větší, pokud naopak tento neuron výsledek nějakým způsobem negativně ovlivňuje výsledek, je jeho váha menší. (11)

Model neuronu

Na obrázku 6 můžeme vidět model neuronu. Do neuronu jsou jako vstup přiváděny výstupy neuronů z předcházející vrstvy. V případě, že se jedná o vstupní vrstvu je vstupem do neuronu námi zadaná hodnota. Neuron provede sumaci všech těchto vstupů vynásobených váhami jejich synapsí a na tuto sumu následně aplikuje aktivační funkci, čímž vytvoří svůj výstup. Tento výstup je následně přiveden na další neuron, pokud se jedná o skrytou vrstvu. V případě, že se jedná o výstupní vrstvu, je tento výstup výstupem sítě. Jako aktivační funkce se používá několik druhů funkcí. Kterou z těchto funkcí použijeme se

rozhoduje podle toho, jakou funkci má síť plnit. Příkladem aktivační funkce může být například Heavisideova funkce. (11)



Obrázek 6 Model neuronu (11)

Trénování neuronových sítí

Proces trénování neuronových sítí je časově velice náročná operace. Během trénování sítě dochází ke snižování rozdílu mezi výstupem sítě a námi požadovaným výsledkem. K tomuto dochází pomocí změny vah, propojujících jednotlivé neurony mezi vrstvami. Nejčastěji se pro tuto operaci používá metoda anglicky zvaná Backpropagation Learning. Pro trénování je potřeba mít k dispozici velké množství vstupních dat, u kterého známe výsledek, který po sítí požadujeme. Jako příklad se můžeme podívat na trénink neuronové sítě pro rozpoznávání objektů. Pokud chceme vycvičit síť na rozpoznání kočky, psa a kola, musíme síti dát vhodné množství fotek koček, psů a kol, a také soubor popisující co na které fotce je. Síť začne s náhodnými váhami a pomocí Backpropagation Learning bude změnami vah zkoušet dosáhnou správného charakterizování příkladu. Síť pokračuje v trénovaní, dokud nejsou všechny příklady správně charakterizovány, nebo dokud nenarazí na podmínku, která ji zastaví. (11)

2.3.2 Deep Learning

Deep Learning je část strojového učení, do které spadají neuronové sítě, které jsou komplexnější než sítě popsané v předchozí kapitole. Jedná se o sítě s větším množstvím skrytých vrstev, větším množstvím neuronů nebo složitější strukturou propojení neuronů.

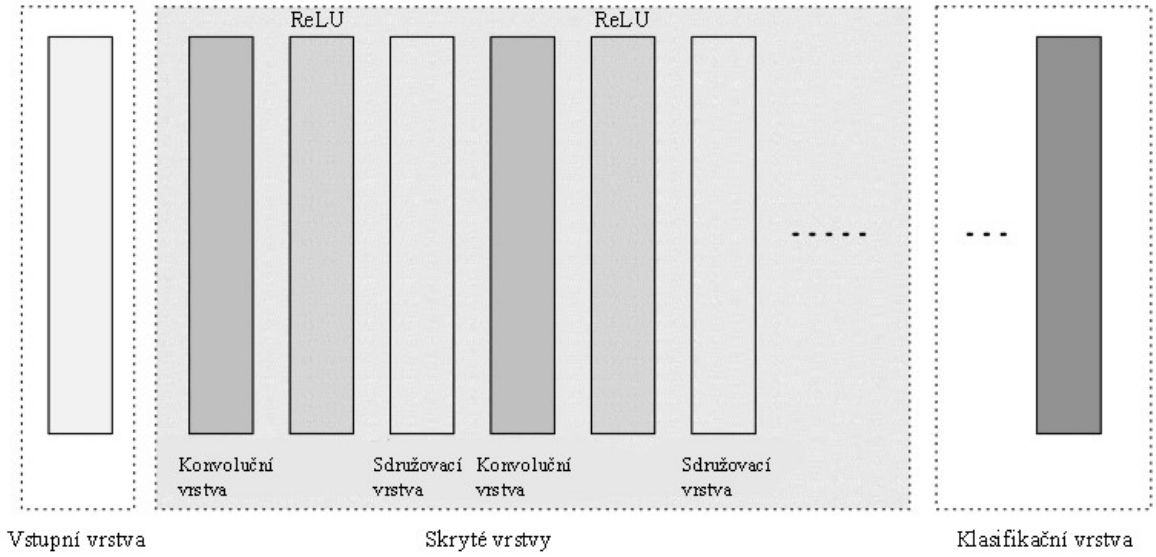
Na rozdíl od toho, co bylo popsáno v předchozí kapitole neuronové sítě spadající do této kategorie nemusí mít všechny neurony spojeny se všemi neurony z předchozích a následujících vrstev. Neurony mohou být spojeny například nejen s neurony z okolních vrstev ale i sami se sebou. Důležitou vlastností je také schopnost těchto sítí samostatně vyhledávat rysy, které jim umožňují klasifikaci objektů. Do této kategorie spadají například konvoluční neuronové sítě, rekursivní neuronové sítě a rekurentní neuronové sítě. Pro názornost se podíváme na konvoluční neuronové sítě. (11)

Konvoluční neuronové sítě

Konvoluční neuronové sítě, označovány zkratkou CNN, jsou sítě, které prokazují nejlepší výsledky v oblasti rozpoznávání obrazu. CNN se používají k rozpoznávání obličejů i objektů. (11)

Architektura

CNN obsahují vstupní vrstvu, výstupní vrstvu a skryté vrstvy. Všechny vrstvy této sítě jsou 3 rozměrné.



Obrázek 7 Architektura konvoluční neuronové sítě (11)

Vstupní vrstva – Vstupní vrstva pro CNN je definována třemi rozměry. Při práci s obrazem tyto rozměry definují výšku a šířku obrazu, třetí rozměr definuje hloubku obrazu, kterou definují hodnoty barev jednotlivých pixelů. BLOB je objekt, který se používá pro ukládání většího množství dat. Používá se pro uložení dat obrazu, který chceme zpracovat. Následně pak tento objekt slouží jako vstup pro neuronovou síť.

Výstupní vrstva – Klasifikační vrstva – Výstupní vrstva nám dává výstup sítě. Všechny neurony této vrstvy jsou propojeny se všemi neurony předchozí vrstvy, ale nejsou připojeny k žádné další vrstvě. Výstupem sítě je matice o rozměrech 1x1xN, kde N je počet tříd, na které je síť trénována. Tato matice obsahuje pravděpodobnosti jednotlivých tříd.

Skryté vrstvy – Jedná se o vrstvy, ve kterých probíhá extrakce rysů při učení. Důležitou vlastností těchto sítí je, že neurony ve skrytých vrstvách nemusí nutně být propojeny se všemi neurony v předchozích vrstvách. Neurony jsou propojeny jen s malým shlukem neuronů v předchozí vrstvě.

Tyto skryté vrstvy obvykle obsahují konvoluční vrstvy a sdružovací vrstvy. (11)

2.3.3 Modul DNN

Blobfromimage(Inputimg, scalefactor, size, mean, swapRB, crop)

Slouží k vytvoření BLOB objektu z obrázku. Parametry funkce:

- *Inputimg* – matice obsahující obraz pro převedení.
- *scalefactor* – umožňuje změnit velikost obrazu.
- *size* – umožňuje určit velikost výstupního obrázku.
- *mean* – umožňuje podělit hloubkovou složku obrázku (RGB) napsanou hodnotou.
- *swapRB* – Bool – Pokud je True, změní barevný kanál z RGB na BGR.
- *crop* – oříznutí obrázku.

readNet(model, config, framework)

Slouží k načtení modelu neuronové sítě. Parametry funkce jsou:

- *model* – odkaz k souboru obsahující váhy pro neurnovou síť.
- *config* – odkaz k textovému souboru obsahující informace o architektuře sítě.
- *framework* – Slouží k určení frameworku, pomocí kterého byla síť vytvořena. Pokud neurčíme framework, tak jej tato funkce rozpozná sama. Podporované frameworky jsou Caffe, Tensorflow, PyTorch a Darknet.

Alternativou k této funkci je použití funkce *readNetFromX*, kde X představuje název frameworku, ve kterém byla síť vytvořena. Funkce *readNet* tyto funkce volá automaticky sama po rozpoznání použitého frameworku.

dnn_network.getUnconnectedOutLayersNames()

Tato funkce vrátí jména výstupních vrstev sítě *dnn_network*.

dnn_network.setInput(imgBlob)

Slouží k nastavení vstupních dat pro síť *dnn_network*.

dnn_network.forward(outputNames)

Slouží k vypočtení výstupů sítě *dnn_network*, ze vstupních dat definovaných ve funkci *setInput*. Parametr *outputNames* obsahuje jména výstupních vrstev sítě, které získáme pomocí funkce *getUnconnectedLayersNames*.

Funkce, které jsou zde popsány, jsou jen některými funkcemi z tohoto modulu. Tyto funkce jsou zde vysvětleny z důvodu jejich následného použití v praktickém případě. (12)

2.3.4 Příklad použití – Rozpoznávání objektů na obrázku

Jako příklad schopnosti OpenCV pracovat s neuronovými sítěmi se zaměříme na rozpoznávání objektů na obrázcích. Cílem bude vytvořit program, který dokáže na předloženém obrázku detekovat a označit objekty, které se na něm nachází. Dále bude program schopen zobrazit informaci, o jaký objekt se jedná a jaká je pravděpodobnost, že je to skutečně daný objekt. V poslední řadě budeme měřit čas potřebný pro dokončení detekce. V závěru následně porovnáme rychlost detekce na platformách PC a Raspberry Pi. Kód příkladu bude vycházet z příkladu dostupném na webu OpenCV.(13)

2.3.5 YOLO – You Only Look Once

Jak bylo zmíněno v úvodu této kapitoly, je pro práci s modulem DNN potřeba mít k dispozici předtrénovanou síť včetně souboru obsahujícího váhy jednotlivých synapsí. V rámci této práce se nebudeme zabývat tvorbou vlastní sítě, a z tohoto důvodu využijeme jednu ze sítí pro detekci objektů. Pro detekci objektů již bylo vyvinuto několik sítí jako například R-CNN, RetinaNET nebo YOLO. Z důvodů, popsaných později v této kapitole bude pro tento příklad použita síť YOLO. Jedná se o konvoluční síť vytvořenou pomocí frameworku Darknet, který je psaný v jazyce C. Tato síť je trénována na COCO data setu. COCO je set dat obsahující fotky různých objektů, pomocí kterých je možné natrénovat síť bez toho, abychom museli sami shromažďovat dostatečně velké množství fotek pro dostatečnou přesnost sítě. Obsahuje 91 kategorií objektů od osob, přes zvířata po dopravní značky a požární hydranty. (14) Pokud bychom chtěli tuto síť natrénovat na práci s naším datasetem, museli bychom získat dostatečně velké množství fotek objektů pro detekci a následně bychom síť trénovali pomocí frameworku Darknet.

Většina sítí vyžaduje, abychom před samotnou detekcí měli již vyhledané oblasti, ve kterých se pravděpodobně nachází nějaké objekty. Tohoto se většinou dosahuje použitím další sítě. Na rozdíl od těchto sítí uplatňuje YOLO jednu síť na celý obrázek, který je

rozdělen na oblasti, a následně určí, ve kterých oblastech se objekty můžou nacházet a přiřadí jim pravděpodobnost, že s v ní nachází daný objekt. Díky tomuto rozdílu je rychlejší než většina ostatních sítí, což se pozitivně projeví například při použití na výkonově omezených zařízeních jako Raspberry Pi nebo při práci s živým feedem.

Samotná síť je k dispozici v několika variantách. K dispozici jsou varianty pro různá rozlišení a také varianta Tiny YOLO, která je ještě rychlejší než klasická YOLO síť. V rámci příkladu využijeme variantu spp. Tato varianta má podobné vlastnosti jako YOLOv3, jediný rozdíl je v nastavení některých vrstev. (15)

2.3.6 Načtení obrázku, převedení do formátu BLOB a základy programu

Jako první je potřeba inicializovat knihovny, které budeme v programu používat. To zahrnuje následující knihovny:

- *cv2* – importuje knihovnu OpenCV pod zkratkou cv.
- *numpy* – importuje knihovnu numpy, která umožňuje práci s maticemi a další matematické operace pod zkratkou np.
- *time* – importuje knihovnu time, kterou použijeme pro měření doby detekce.

Následně deklarujeme proměnné, které budeme používat:

- *img* – obrázek, na kterém bude prováděna detekce.
- *inputimg* – za použití funkce *imread*, převedeme vstupní obrázek na pole pixelů.
- *model* – slouží k načtení vah synapsí pro neuronovou síť.
- *config* – slouží k načtení souboru obsahující informace o architektuře neuronové sítě.
- *font* – proměnná obsahující definici fontu, kterou budeme používat pro vypsání textu.
- *confThreshold* – parametr, kterým určujeme požadovanou minimální pravděpodobnost, se kterou se jedná o daný objekt.
- *blobimg* – proměnná obsahující obrázek, který byl pomocí funkce *cvtColor* převeden z barevného formátu RGBA (Red, Green, Blue, Alpha) do formátu RGB.
- *imgBlob* – obsahuje obrázek uložený ve formátu objektu BLOB, který je následně použit pro detekci v síti.
- *resized_img* – obsahuje obrázek převedený do velikosti 800x800 pixelů.

```
import cv2 as cv
import numpy as np
import time

img = "./cat_side.jpg"
inputimg = cv.imread(img)
model ="./yolov3-spp.weights"
config = "./yolov3-spp.cfg"
font = cv.FONT_HERSHEY_SIMPLEX
confThreshold = 0.5
blobimg = cv.cvtColor(inputimg, cv.COLOR_RGBA2RGB)
imgBlob = cv.dnn.blobFromImage(blobimg, 1/255, size=(800,800))
resized_img = cv.resize(inputimg, (800,800))
classesstr = "person,bicycle,car,motorbike,aeroplane,bus,train,truck,boat,traffic light,fire hydrant,
classes = classesstr.split(",")
```

Obrázek 8 Základní část programu

2.3.7 Detekce

Tato část kódu plní několik funkcí. V první části probíhá nastavení neuronové sítě a detekce. Model sítě je uložen v proměnné s názvem *dnn_network*. Následně získáme jména výstupních vrstev sítě a nastavíme její vstup. Nakonec pomocí funkce *forward* provede detekci.

Další část kódu slouží k zjištění souřadnic boxů obsahující detekované objekty a jejich pravděpodobností. Pro každou detekci ve výstupu získáme pravděpodobnosti jednotlivých tříd a pomocí funkce *argmax* zjistíme, pro kterou třídu je pravděpodobnost největší. Pokud tato pravděpodobnost přesahuje námi nastavenou minimální mez, vypočteme souřadnice hranic boxu, ve kterém se nachází a uložíme její ID do proměnné *classIds*. V poslední části těmto třídám přiřadíme jméno ze seznamu uloženého v proměnné *classes*.

```
set_time = time.time() #start detekce

dnn_network = cv.dnn.readNet( model, config)
print("Network loaded")
outNames = dnn_network.getUnconnectedOutLayersNames()
dnn_network.setInput(imgBlob)
print("Input set")
retval = dnn_network.forward(outNames)
print("Detection finished")

end_time = time.time() #konec detekce
detection_time = round(end_time-set_time,3)
print("Detekce trvala ", detection_time, "sekund.")

classIds = []
confidences = []
boxes = []
names = []

for out in retval:
            for detection in out:
                scores = detection[5:]
                classId = np.argmax(scores)
                confidence = scores[classId]
                if confidence > confThreshold:
                    center_x = int(detection[0] * 800)
                    center_y = int(detection[1] * 800)
                    width = int(detection[2] * 800)
                    height = int(detection[3] * 800)
                    left = int(center_x - width / 2)
                    top = int(center_y - height / 2)
                    classIds.append(classId)
                    confidences.append(float(confidence))
                    boxes.append([left, top, width, height])

for x in range(len(classIds)):
    names.append([classes[classIds[x]]])
```

Obrázek 9 Kód detekce

2.3.8 Vizualizace výsledků detekce

Poslední částí příkladu je grafická prezentace detekce. Pomocí smyčky *for* vykreslíme obdélníky pro všechny detekované objekty v proměnné *boxes* a pomocí funkce *putText* vypíšeme název třídy a pravděpodobnost, se kterou se jedná o daný objekt.

```
for i in range(len(boxes)):
    cv.rectangle(resized_img,(boxes[i][0], boxes[i][1]),
    (boxes[i][0] + boxes[i][2], boxes[i][1]+boxes[i][3]), (0, 0, 255), 2)
    textsize, base = cv.getTextSize(names[i][0], font,2,1)
    textsizeconf, base = cv.getTextSize(str(round(confidences[i],3)), font,1,1)
    cv.rectangle(resized_img, (boxes[i][0], boxes[i][1]),
    (boxes[i][0] + boxes[i][2], boxes[i][1]-textsize[1]), (0,0,255),cv.FILLED)
    cv.putText(resized_img, names[i][0], (boxes[i][0], boxes[i][1]-5), font,1,
    (255,255,255), 1)
    cv.putText(resized_img, str(round(confidences[i],3)), (boxes[i][0] + boxes[i][2]-textsizeconf[0],
    boxes[i][1]-5),font,1, (255,255,255), 1)

cv.imshow("out", resized_img)
cv.waitKey(0)
```

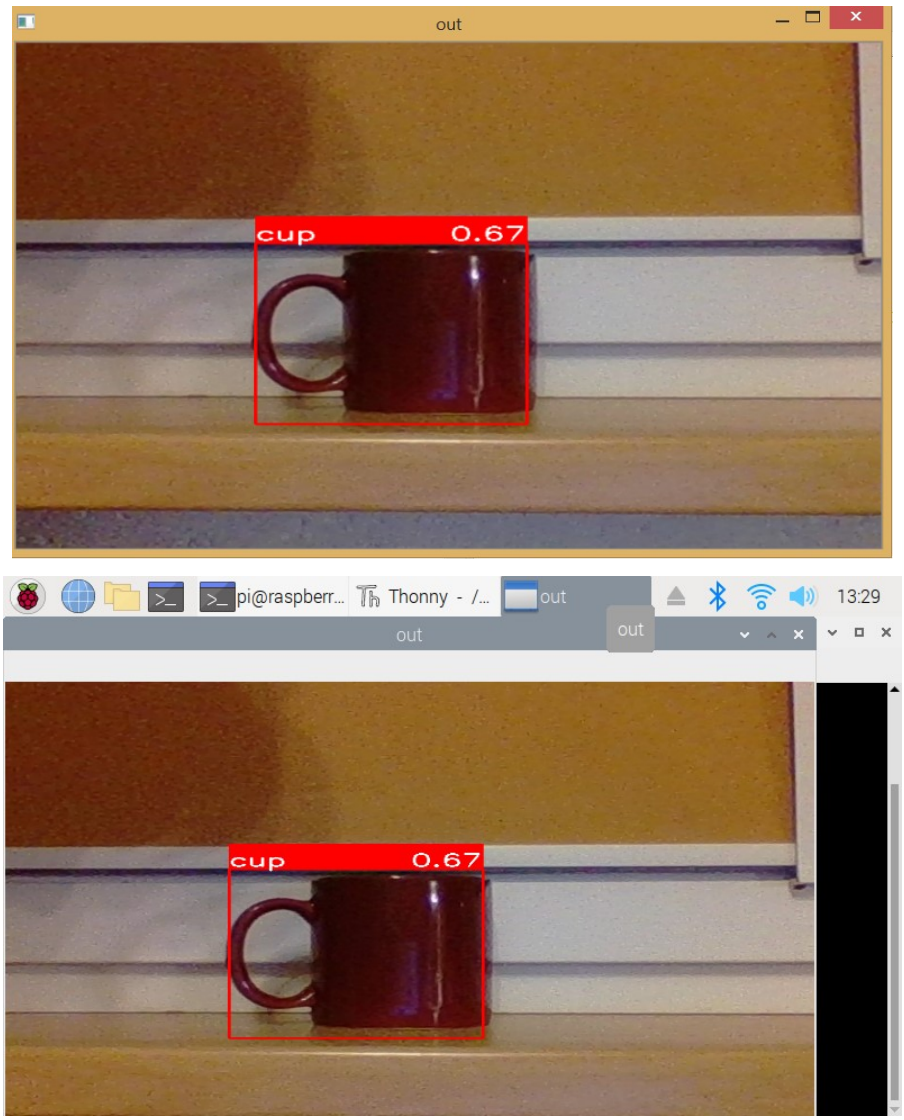
Obrázek 10 Kód vykreslení výsledků

2.3.9 Porovnání doby detekce na platformách PC a Raspberry Pi

Rozdílem při použití neuronových sítí při detekci na různých platformách bude doba, jakou bude trvat. Tento čas závisí na výpočetním výkonu daného zařízení. Při použití stejných fotek můžeme porovnat časy detekce na Raspberry Pi a na PC. Z tabulky je vidět, že tento rozdíl je znatelný a při práci s živým feedem bude pravděpodobně způsobovat problémy.

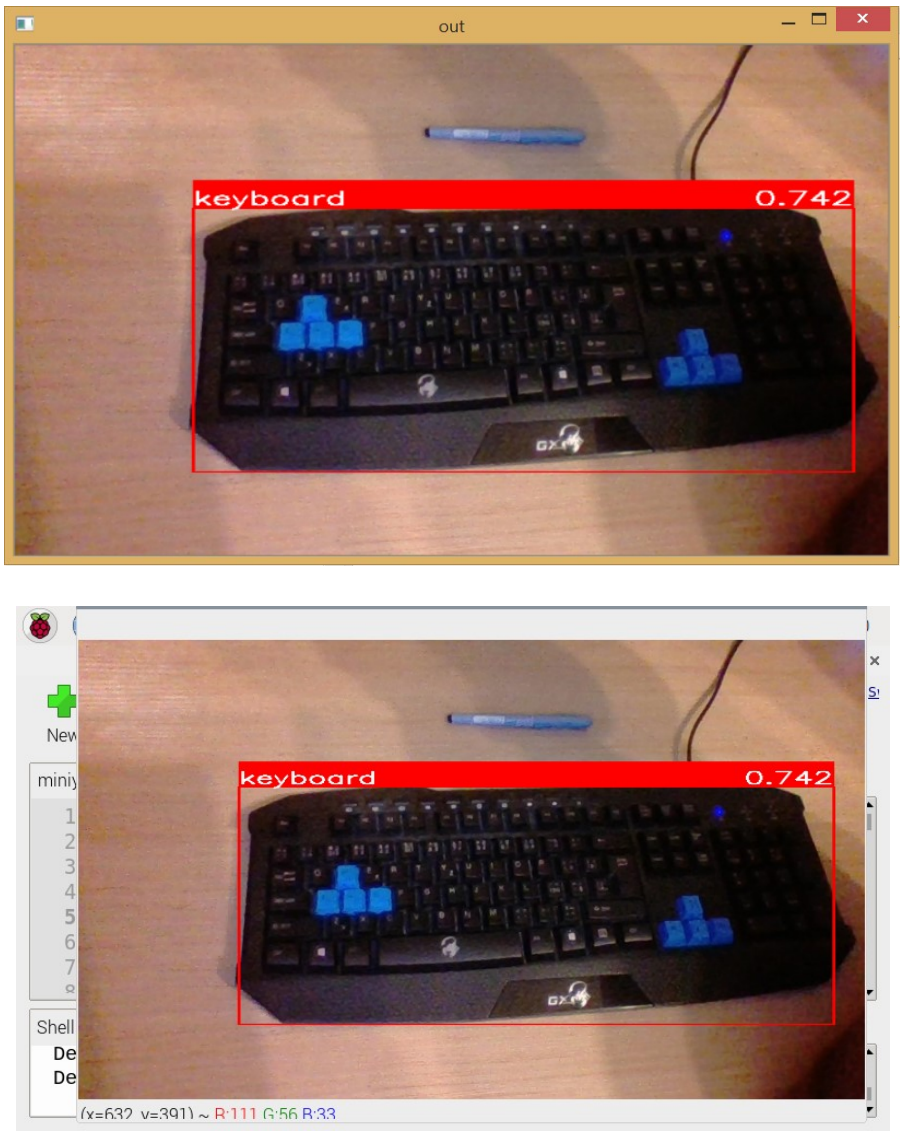
Tabulka 1 Časy detekce

Fotka	Čas Raspberry Pi 3 [s]	Čas PC [s]
Hrnek	73,931	10,238
Klávesnice	71,6	10,039



Obrázek 11 Detekce hrnku na PC a Raspberry Pi(dole)

Jako příklad objektů k detekci použijeme 2 objekty obsažené v COCO setu. Jedním z těchto objektů je hrnek a druhým klávesnice. Je vidět, že i přes relativně nízkou kvalitu fotografií je síť schopna rozpoznat objekty, které se na ní nacházejí. Pro použití programu na platformě Raspberry nebylo potřeba nijak upravovat program, během detekce ale docházelo k znatelnému zahřívání desky.



Obrázek 12 Detekce klávesnice na PC a Raspberry Pi(dole)

2.3.10 Detekce na feedu z webkamery a porovnání platforem

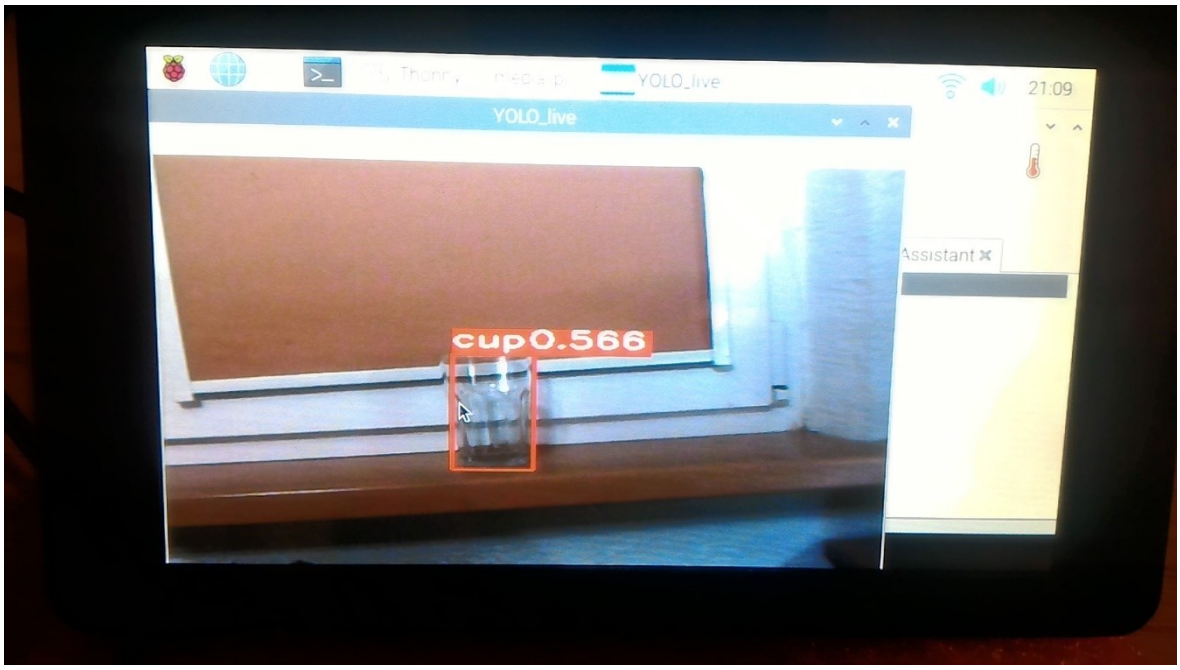
Cílem této práce je ukázat možnosti knihovny OpenCV při práci s videem snímaným z webkamery. Snímání videa z webkamery v této knihovně probíhá zachycením jednoho snímku a jeho zobrazení. Toto provádíme ve smyčce, čímž dosáhneme vytvoření video streamu.

Z tabulky 1 je patrné, že na platformě Raspberry Pi 3 může trvat zpracování jednoho snímku i déle než 1 minutu. To je pro práci s živým obrazem příliš dlouhá doba. Z tohoto důvodu místo *spp* verze sítě YOLO využijeme verzi *Tiny YOLO*. Tato verze je výrazně rychlejší, ale je také méně přesná. Za použití této sítě a detekce na přenosu z webkamery porovnáme schopnosti platforem PC, Raspberry Pi 3 a 4, při práci s takto výpočetně náročnou úlohou. Kód, který byl pro tuto úlohu vytvořen je k dispozici v příloze A. Použité příkazy jsou zčásti vysvětleny v předešlé kapitole. Funkce pro práci s živým přenosem

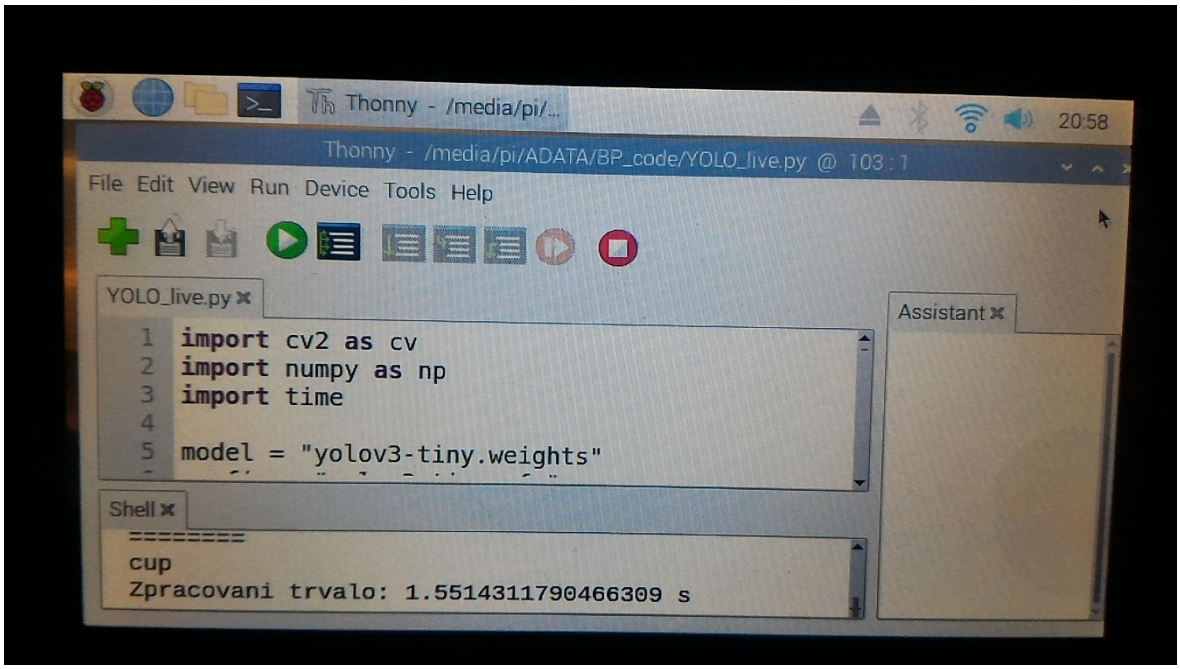
budou vysvětleny v následující kapitole. Průměrné detekční časy pro jednotlivé platformy můžeme vidět v tabulce 2.

Tabulka 2 Časy pro zpracování snímku na daných platformách za použití tiny YOLO sítě

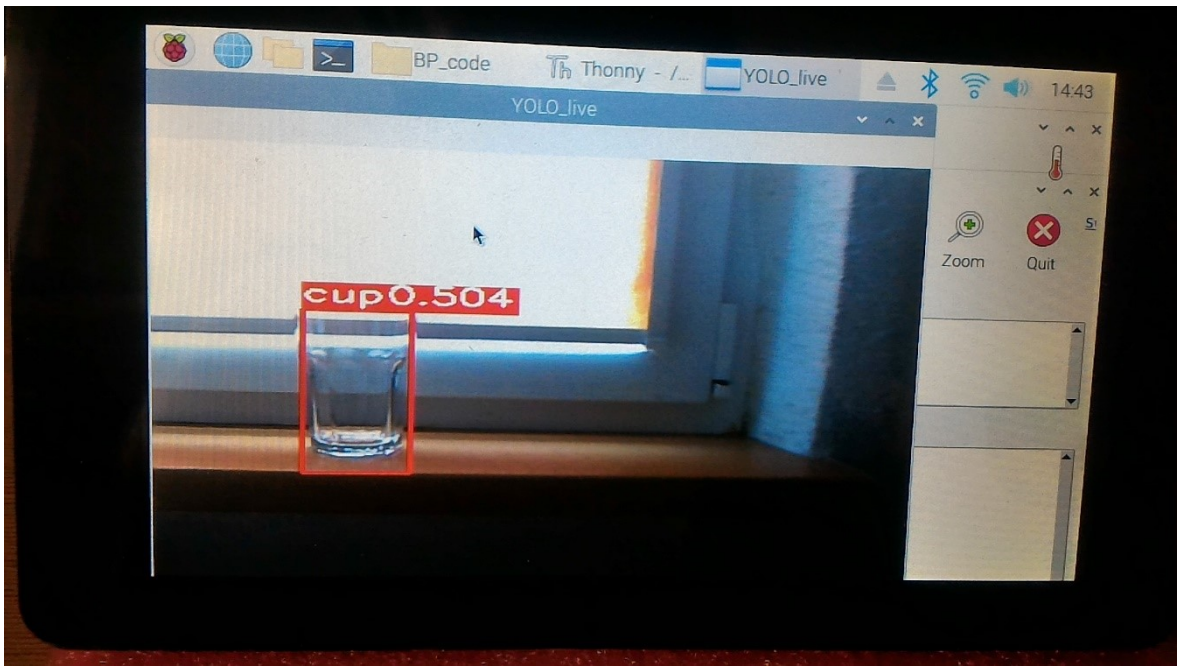
Platforma	**PC**	**Raspberry Pi 3**	**Raspberry Pi 4**
Čas [s]	0,08	1,6 až 2	0,55 až 0,65



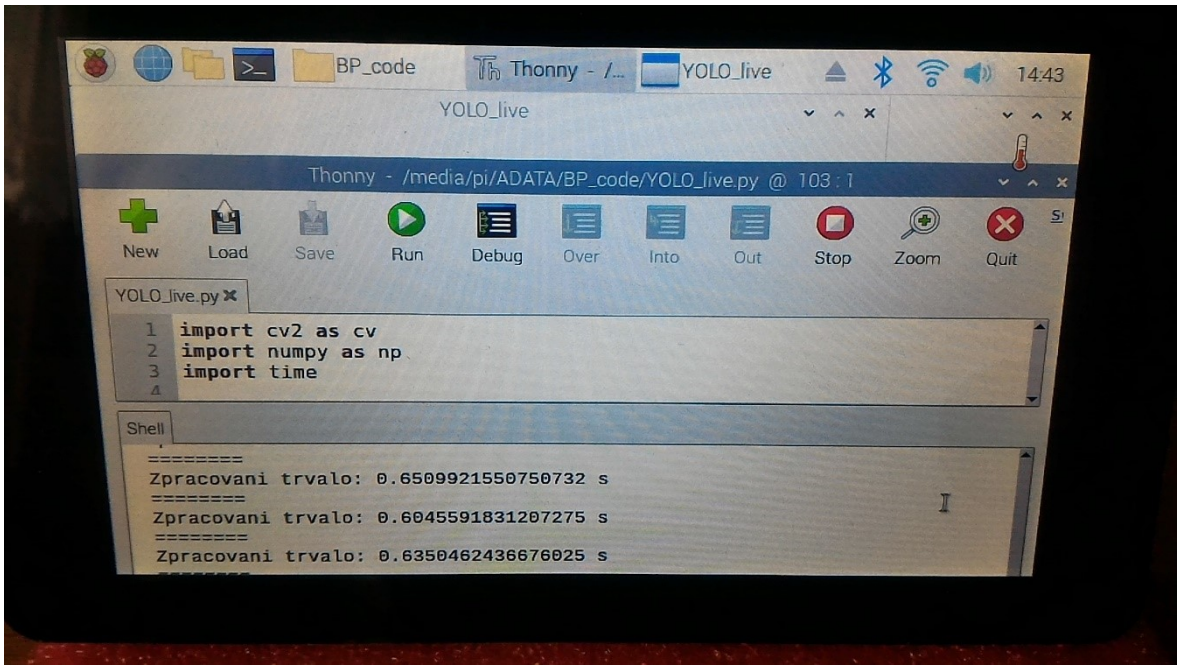
Obrázek 13 Detekce na přenosu z webkamery na RPi 3



Obrázek 14 IDE pro python na platformě RPi 3



Obrázek 15 Detekce na přenosu z webkamery na RPi 4



Obrázek 16 IDE pro python na platformě RPi 4

Z časů je vidět, že platforma RPi výkonově není optimální pro práci s živým přenosem. Při zobrazování je patrné zasekávání a při vzniklém časovém zpoždění se stává, že se nám na výstupu vykresluje něco, co už před kamerou neleží. Platforma RPi 4 na toto trpí méně, je zde patrný rozdíl v dobách zpracování snímků. I přes tento fakt ale tento jev nezmizel úplně. Pro snímání obrazu na platformách RPi i PC byla použitá stejná USB kamera.

3 OpenCV a kontinuální snímání z webkamery

Další příklad bude zaměřen na schopnost knihovny OpenCV pracovat s živým přenosem získaným z webkamery. Tato úloha se bude skládat ze dvou částí. První částí bude pořízení a zobrazení snímku z webkamery. Druhou částí bude zpracování snímku před jeho zobrazením. Jako zpracování snímku budeme uvažovat úpravy, které je nutné provést pro použití snímku jako vstup do neuronové sítě, a především detekci a klasifikaci obličejů, které se na snímku vyskytnou. Prakticky si tuto úlohu můžeme představit jako velice jednoduchý bezpečnostní monitorovací systém, kdy obsluha kontroluje, který zaměstnanec prošel určitým bezpečnostním checkpointem (např. vrátnicí). Budeme tedy chtít, aby se nám na video streamu objevilo jméno zaměstnance. Systém disponující těmito bychom mohli nazvat velice jednoduchým systémem augmentované (rozšířené) reality. Vytvořený kód pro tuto úlohu je v příloze C.

3.1 Snímání a zobrazování

VideoCapture()

Pro snímání obrazu z webkamery musíme nejprve vytvořit objekt, určující, kterou kameru má program použít. K tomuto účelů slouží funkce *VideoCapture()*. Stejně jako spousta jiných funkcí v knihovně OpenCV má i tato funkce několik možných variací. Naše varianta bere jako argument funkce identifikátor kamery, kterou chceme použít. Pro automatické určení můžeme jako argument využít hodnotu 0, která otevře výchozí kameru systému. Další varianty této funkce umožňují například načítání video souboru. (16)

Následně pro získání snímku z kamery využijeme funkci *read()* na vytvořený objekt obsahující informace o kameře.

K zobrazení snímku využijeme funkci *imshow()*, která již byla popsána dříve. Abychom zajistili, že snímání a zobrazování bude probíhat neustále, a my tak získávali kontinuální sled obrazů, bude snímání a zobrazování vnořeno do smyčky *while*. Následně je nutné, aby v této smyčce byl i příkaz *waitKey()*, aby se zobrazovací okno nezavíralo. (13)

3.2 Detekce a klasifikace

Další částí této úlohy je detekce a klasifikace obličejů. Detekcí rozumíme proces, při kterém zjišťujeme, zda a kde se obličej nachází na aktuálním snímku. K tomuto cíli lze využít buďto neuronovou síť nebo například HAAR kaskády. Klasifikací následně určíme, o koho se jedná. Za tímto účelem využijeme neuronovou síť. My pro obě tyto operace využijeme neuronové sítě.

3.2.1 Facenet-pytorch

Abychom nemuseli trénovat a vytvářet neuronovou síť sami, využijeme knihovny Pytorch a Facenet_pytorch. Knihovna pytorch umožňuje vytvářet a trénovat neuronové sítě. Obsahuje také knihovnu torchvision, která sama obsahuje několik typů neuronových sítí. My ale tuto knihovnu využijeme jenom pro zajištění funkčnosti knihovny Facenet_pytorch. Facenet_pytorch je knihovna, obsahující předtrénovanou neuronovou síť facenet (v této knihovně reprezentována sítí pojmenovanou InceptionResnet), která, jak napovídá název, slouží k identifikací obličejů. Síť facenet obsažena v této knihovně umožňuje jak klasifikaci obličejů (určení o koho se jedná), tak vytváření tzv. face embeddings, které lze následně ke klasifikaci použít. Tato knihovna obsahuje také trénovanou síť MTCNN, která slouží k detekci obličejů. Facenet je připraven pro použití s knihovnou pro práci s obrazem PIL, která na rozdíl od OpenCV využívá k ukládání obrazů tenzory namísto matic. (17)

MTCNN

MTCNN představuje jednu ze sítí, které nám knihovna facenet_pytorch poskytuje. Jedná se o síť natrénovanou k detekci obličejů. Defaultně je funkce pro tuto síť nastavena tak, že na fotce vyhledá obličej, ořízne jej a uloží do tenzoru pro další zpracování. Tato možnost je vhodná pro práci při klasifikaci fotografií jedné osoby, případně při zpracování dat pro trénování sítě pro rozpoznání obličeje. Pokud ale chceme na snímku najít více obličejů a rozpoznat je, jako je tomu v našem případě, je tento přístup nedostačující.

Pro naše účely potřebujeme, aby síť našla všechny obličeje. Pro tuto eventualitu disponuje MTCNN atributem *keep_all*, který zaručuje, že síť uchová všechny nalezené obličeje. Jejím výstupem je následně vícerozměrný tenzor obsahující ořezané obličeje. Tato možnost nám neřekne souřadnice těchto obličejů. Pro jejich získání proto použijeme síť MTCNN ještě jednou, tentokrát s doplňujícím parametrem *eval()*. V této konfiguraci nám síť vrátí souřadnice hraničních bodů obličejů a pravděpodobnost, že se jedná o obličej.

(17)

Jak bylo dříve zmíněno pracuje síť MTCNN s knihovnou PIL a jako vstup tedy očekává pytorch tenzor. To může vyvolat problém při nastavování vstupu. Knihovna OpenCV při uložení snímku z webkamery využije formát numpy array, který se od tenzorů liší. Pokud se pokusíme převést snímek pomocí funkce pro převod, může se stát, že při konverzi vznikne formát tenzoru, který pro MTCNN bude indikovat že se jedná o batch (várku) obrazů, nikoliv o jeden snímek. Proto je zde vhodnějším řešením převést snímek pořízen pomocí OpenCV do tenzorového formátu knihovny PIL.

InceptionResnet

K samotné identifikaci budeme využívat síť *InceptionResnet.* Pokud síť před použitím nijak nenastavujeme, slouží k vytvoření face embeddings z fotografií. Pokud ji ale chceme využít pro klasifikaci, musíme ji nastavit do módu *eval().* Pokud je síť použita pro klasifikaci, jejím výstupem bude pole obsahující tzv. logits pro jednotlivé třídy, na které je trénována. (17)

Logit je definován vztahem:

$f(x) = log \frac{x}{1-x}$, (18)

kde x reprezentuje námi hledanou pravděpodobnost. Pro získání pravděpodobností je proto potřeba výsledky převést. Knihovna nám umožňuje použít předtrénovanou síť, čehož využijeme při trénování sítě pro náš dataset.

Trénování sítě Facenet

Pro trénování sítě Facenet využijeme kód, který je dostupný v github repositáři knihovny (18). Síť natrénujeme na námi určený počet tříd, každá třída bude definována co největším počtem fotek. V našem datasetu některé třídy přesahují i množství 70 fotografií, jiné však pracují například s 30 fotografiemi. Tyta čísla nejsou ničím dána, sami si určujeme, kolik fotografií chceme použít.

Jedním důležitým aspektem je dodržení struktury složek. Jako vstup zadáváme do kódu složku obsahující podsložky s fotografiemi lidí k rozpoznání. Využívá se struktura, při které se každá složka jmenuje podle třídy, kterou reprezentuje. Pomocí toho pak síť pojmenuje jednotlivé třídy podle názvů složek. (20)

Jedním námi voleným parametrem je tzv. batch size. Jedná se o množství vstupních dat pro jeden výpočet sítě. Například pokud budeme mít dataset obsahující 8 fotek a batch size bude roven 4, pak na jeden průběh sítí pošleme 4 fotky. Celý proces potom bude mít pouze

2 iterace. Dalším parametrem je počet epoch. Počet epoch určuje, kolikrát má celý dataset projít sítí. Jedna epocha se skládá z dopředného průchodu (v našem případě výpočet pravděpodobností) a zpětného chodu. Zpětný chod slouží ke zpětné propagaci chyb a přepočtení vah. Pokud máme malý dataset umožní nám větší počet epoch získat lepší přesnost sítě. (21)

V kódu můžeme také vidět, že se kontroluje, jestli máme k dispozici grafickou kartu s CUDA procesory, které jsou vhodné pro výpočet. Pokud takovou grafickou kartu k dispozici nemáme, používá se k tréninku CPU. CUDA procesory dosahují při potřebných výpočtech znatelně lepších časů. Je také vhodné zmínit, že přesnost detekce na našem datasetu není příliš velká. Z části se pravděpodobně jedná o důsledek toho, že náš dataset není příliš rozsáhlý.

3.2.2 Detekce a získání pravděpodobností tříd, vizualizace

Po získání snímku z webkamery následuje proces, při kterém na snímku najdeme obličeje, ořežeme snímek tak, aby obsahoval pouze jeden obličej, a následně získáme jednotlivé pravděpodobnosti tříd. V režimu vyhodnocení je vstupem do sítě pouze jeden snímek. Tento snímek má být oříznut tak, aby obsahoval pouze jeden obličej, měl velikost 160x160 pixelů a byl tzv. vybělen („Whitened“). (16) Z tohoto důvodu bude celá detekce a klasifikace probíhat ve smyčce o tolika iteracích, kolik je na snímku nalezeno obličejů. Tím se vyvarujeme zbytečně velkého množství proměnných a zaručíme, že projdeme opravdu všechny nalezené obličeje. Protože sítě pracují s tenzory a nepodporují práci s maticemi numpy, které využívá OpenCV, převedeme pořízené snímky pokaždé do formátu knihovny *PIL,* pomocí funkce *Image.fromarray.* (22)

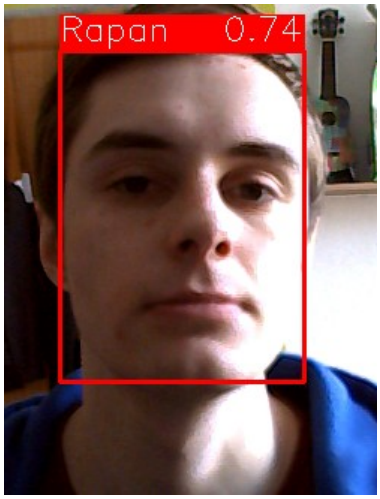
Jako první získáme pomocí funkce *mtcnn.detect()* tenzor obsahující boxy určujících souřadnice obličejů a pravděpodobnosti, že se jedná o obličeje. Následně potom získáme vektor oříznutých obličejů *faces* pomocí funkce *mtcnn.*

Pomocí smyčky *for* následně budeme iterovat přes celé pole obsahující data o obličejích *faces*, čímž docílíme toho, že obličeje zpracováváme po jednom. Zároveň budeme pomocí proměnné *i* sledovat kolikátý obličej v pořadí zrovna identifikujeme. Vektor právě rozpoznávaného obličeje převedeme do tvaru, kdy první číslice vektoru určuje počet snímků v daném batchi, tedy kolik snímku se v proměnné objeví, pomocí funkce *unsqueeze.* V našem případě by tedy měla být velikost batche 1 snímek. Následně pomocí sítě resnet provedeme samotnou identifikaci. Při ní získáme a uložíme do proměnné *person* informace o hodnotách pravděpodobností jednotlivých tříd. Proměnnou *person* poté převedeme

do maticového tvaru pro zjednodušení práce s ní. Dále zjistíme, které ze tříd patří největší pravděpodobnost, a tím zjistíme, o jakou osobu se jedná. Posledním krokem je vložení jména osoby a pravděpodobnosti, s jakou se o tuto třídu jedná, do snímku a jeho vykreslení.

3.2.3 Grafické zobrazení výsledků

Cílem této úlohy je, aby pozorovatel, který sleduje stream z webkamery, viděl, kdo se v daný okamžik na záběru nachází. Informace o pravděpodobnosti, že se o danou osobu jsou zajímavé více pro nás jako ukazatel kvality detekce než pro uživatele. I přesto ji ale naznačíme do snímku.



Obrázek 17 Vizualizace výsledků detekce

Do snímku tedy bude vnášet text obsahující jméno osoby a pravděpodobnost shody. Vložíme také rámeček, ohraničující obličej osoby. Pro lepší čitelnost bude i text orámován.

3.3 Rychlost detekce

Je vhodné se zmínit o tom, že tento způsob detekce a identifikace osob je relativně pomalý. Je proto velice pravděpodobné, že zprovoznění tohoto modelu na platformě Raspberry Pi 3 nebude možné, z důvodů omezeného výpočetního výkonu. Důvodem je velice pravděpodobně skutečnost, že k celému procesu využíváme 2 neuronové sítě, které nejsou dále nijak optimalizované, jsou pouze použity ve stavu, v jakém jsme je získali z jejich repositáře. Řešením by mohlo být buďto provedení optimalizace sítí, případně použití jiných sítí, nebo například využití platformy RPi pouze k zobrazení a snímání obrazu. V tomto případě bychom mohli využít komunikaci RPi se serverem, případně i jiným PC. Toto PC se větším výpočetním výkonem by mohlo detekci zvládat podstatně rychleji a v závislosti na rychlost připojení by následně celkový proces mohl být rychlejší.

Další variantou pro dosažení rychlejších výpočtů je provádět tyto výpočty na počítačích disponujících grafickou kartou s CUDA procesory. Jak již bylo zmíněno, dosáhneme tím znatelně menších výpočetních časů. Aby se této vlastnosti dalo skutečně využít je třeba zasahovat do konfigurací sítí. Kód, který je k dispozici na github repositáři sice umožňuje využití CUDA procesorů. Během průchodu kódem ale nastal problém, kdy síť očekávala, že bude pracovat právě s CUDA procesorem, ale nedošlo k přesunutí vstupního tenzoru snímku do tohoto procesoru, a tak nastala chyba. Pokud bychom tedy zasahovali do architektury sítě, mohli bychom dosáhnout lepších výsledků. Na PC disponující procesorem AMD Ryzen 5, tedy 6 jádrovým procesorem o frekvenci 3,2GHz, trvá zpracování jednoho snímku zhruba 0,9 sekundy.

3.4 Rozšíření funkcí současného systému

V úvodu tohoto příkladu jsme jej popsali jako velice jednoduchou augmentovanou realitu. Na monitoru nám běží stream videa ze skutečného světa, doplněný o informace o osobách, které se na něm vyskytují. Pokud bychom tento program opravdu využívali pro kontrolu v bezpečnostních checkpointech, mohly by pro nás být užitečné i další informace o subjektech na videu. Řekněme, že bychom například chtěli vidět i věk osob, jejich zaměstnanecké ID, pozici ve firmě, případně nějaké další poznámky. Mohli bychom také chtít vědět, kdy naposledy tato osoba prošla tímto checkpointem. Implementaci těchto funkcí dosáhneme využitím základních modulů *os*, *time* a *datetime*. Samotného vložení informací na snímek opět dosáhneme pomocí OpenCV.

3.4.1 Implementace údajů o subjektech

Budeme uvažovat firemní využití tohoto softwaru. Můžeme tedy předpokládat, že máme k dispozici údaje o subjektech (v tomto případě zaměstnancích), které by měl dozor vidět. Prakticky by se většina informací pravděpodobně ukládala do databází a jejich vyhledávání by se provádělo pomocí SQL dotazů. Abychom nemuseli projekt rozšiřovat navíc i o SQL část, budeme data pro zobrazení ukládat jako textový soubor, ze kterého budeme následně číst. Ten bude formátován tak, že jednotlivé řádky toho, co chceme zobrazit budou vždy odděleny pomocí středníku.

```
Rapan;Age:;TBD;ID:;RAP0015;Student
```

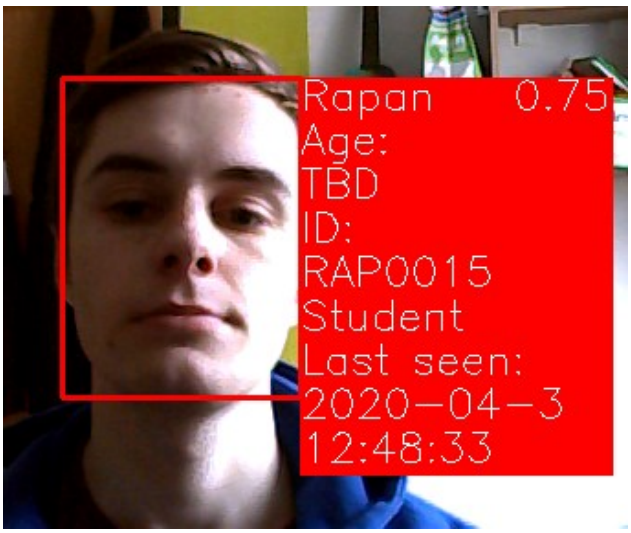
Obrázek 18 Formát textového souboru

Tyto soubory budou pojmenovány jako „*id_subjektu.txt“*. Celý proces získání a vykreslení těchto informací zajišťuje funkce *drawData()*. Jejími vstupy budou *id*, což je jméno subjektu, *prob*, tedy pravděpodobnost shody, *pic*, který definuje, na jaký snímek se

budou data vykreslovat a *facearr*, což je pole obsahující informace o souřadnicích boxů ohraničujících obličeje. Posledním parametrem potom bude *last_seenshow*, který určuje čas, kdy byl subjekt naposledy viděn.

Jedna část této funkce vykreslí hranice obličeje a obdélník, který bude obsahovat informace. Aby se vešel celý text informací do obdélníku, zjišťujeme, která část textů je nejdelší a podle té určíme i šířku obdélníku. Tento obdélník bude obsahovat i informace o tom, kdy se subjekt naposledy objevil, je tedy třeba prodloužit jeho výšku o 3 řádky.

Tato funkce také otevře soubor s informacemi subjektu, načte jeho obsah a následně pomocí funkce *split* rozdělí textový řetězec obsahují celý text souboru. Tento rozdělený řetězec se uloží do pole textových proměnných, kterým následně iterujeme a vypisujeme každý řádek zvlášť na jinou pozici na snímku. Vykreslování textu na řádky pod sebou dosáhneme průběžným sčítáním výšky řádků. Dochází v ní také k vypsání pravděpodobnosti shody.



Obrázek 19 Vizualizace výsledků detekce s daty o subjektu

3.4.2 Časové informace

Další funkcí, kterou vytvoříme bude funkce *TimeFunction()*. Jejími parametry budou *current_time*, tedy současný čas ve formátu *datetime, timedir*, což je složka, ve které budou umístěny záznamy toho, kdy byl subjekt spatřen, v textovém souboru, *id*, které definuje jméno subjektu a *timeinsec*, což je současný čas v sekundách. Na začátku této funkce otevřeme soubor, obsahující časové údaje, v režimu čtení. Následně z něj načteme potřebné údaje a soubor zavřeme. Tato funkce bude vracet zpátky textovou proměnnou obsahující

informace o posledním čase, kdy byl subjekt spatřen, ve formátu *datetime*. Vrací tedy čas k zobrazení na snímku.

Náplní této funkce bude kontrola toho, kdy se rozpoznaný subjekt naposledy objevil. V textovém souboru, který tyto informace bude obsahovat se budou vyskytovat dva druhy časů. Oba dva tyto údaje reprezentují stejný časový okamžik. První z nich je čas ve formátu *datetime*, který poskytuje knihovna *datetime*. Tento čas je zapsán ve formátu *YYYY-MM-DD Hodina:minuta:sekunda*. (23) Tohoto údaje využijeme pro informaci pozorovatele, ale pro porovnání se současným časem je kvůli jeho formátu nevhodný. Slouží tedy pouze k vykreslení do obrazu. Druhý čas je čas reprezentován v sekundách od začátku času pro daný OS. (24) Tento čas budeme využívat pro porovnání současného času a času, kdy byl subjekt naposledy spatřen. Tyto dva časy od sebe budou odděleny pomocí čárek, abychom je následně mohli rozdělit pomocí funkce *split* a uložit je do dvou rozdílných proměnných. Časové údaje o dvou různých instancích spatření subjektu budou odděleny pomocí středníku, abychom je mohli rozdělit a číst poslední čas, kdy byl subjekt spatřen.

```
2020-04-25 21:49:18.612183,1587844158.612183
```

Obrázek 20 Formát uloženého textu obsahující časové informace

Samotné porovnání realizujeme pomocí *if* podmínek. První zkontrolujeme, jestli není proměnná obsahující čas v sekundách prázdná. Pokud prázdná je, určíme čas k zobrazení jako současný čas. Následně otevřeme soubor s informacemi o čase v režimu *append*. K souboru pak připneme současný čas ve formátu *datetime* a za čárkou čas v sekundách.

Pokud první tato proměnná není prázdná, znamená to, že už byl subjekt spatřen. V tomto případě zkontrolujeme, jestli je rozdíl v současném čase v sekundách a v posledním zapsaném čase v sekundách větší než 60 sekund. Pokud ano, otevřeme soubor v *append* režimu a připíšeme informace o nových časech a přiřadíme nový čas pro zobrazení. Pokud rozdíl není větší než 60 sekund, pak je čas pro zobrazení čas, který načteme z textového souboru.

Závěr

Tato práce se věnovala použití knihovny OpenCV v jazyce Python a ukázala převážně její možnosti v oblasti počítačového vidění za podpory neuronové sítě. První kapitola je věnována jazyku Python. Jedná se o jazyk oblíbený jak mezi profesionálními programátory, tak mezi nadšenci do IT. Mezi jeho hlavní přednosti patří hlavně jednoduchost a srozumitelnost jeho kódu a díky širokému spektru dostupných knihoven jde použít k vývoji různých aplikací od počítačových her, po vědecké programy. Jeho použití se na různých OS nijak zásadně neliší, jediný rozdíl je z pravidla v instalaci a knihovnách, které daný OS podporuje. Je zde také popsána platforma Raspberry Pi. Při této práci byl pro ukázku použit model Raspberry Pi 3 Model B a Raspberry Pi 4 Model B, jejichž hardwarové specifikace jsou popsány v kapitole 1.4. Jako operační systém tato platforma může využít různých OS založených na Linuxu od Ubuntu po Kali Linux. Oficiálně je však doporučeným OS Raspbian, založený na Linuxové distibuci Debian, který je dostupný na webových stránkách výrobce. Operační systém se na platformách Raspberry může bootovat pomocí microSD karty obsahující systém, nebo pomocí USB disku.

Druhá kapitola této práce je zaměřená na samotnou knihovnu OpenCV. Jedná se o knihovnu zaměřenou na počítačové vidění. Její možnosti ale obsahují i možnosti práce se strojovým učením, neuronovými sítěmi a také se sítěmi spadajícími do kategorie Deep Learning. V další části této kapitoly jsou popsány některé funkce pro zpracování obrazu a práci s neuronovými sítěmi použitými v následném příkladu. Poslední část této kapitoly je věnována příkladům použití této knihovny. Pro ukázku možností této knihovny při práci s neuronovými sítěmi, spadajícími pod Deep Learning, je jako příklad použita detekce objektů na fotografiích pořízených pomocí webkamery. Pro detekci je použita neuronová síť YOLO – You Only Look Once, která byla tvůrci vycvičena na data setu COCO. Závěrem této části je porovnání rychlostí detekcí na platformách PC a Raspberry Pi, ze kterých je patrné, že doba detekce je přímo závislá na výpočetním výkonu. Z tohoto důvodu trvá detekce na Raspberry cca 70 sekund, kdežto na PC tato detekce trvá zhruba 10-15 sekund. Porovnali jsme také dobu zpracování jednoho snímku na platformách PC, RPi 3 a 4, při práci s živým přenosem. Na těchto dobách bylo patrné, že platforma RPi nemá dostatečný výkon pro použití neuronové sítě YOLO k monitorování objektů na živém přenosu.

Poslední kapitola této práce je věnována vytvoření příkladu pro praktické využití této knihovny, za využití webkamery k získání živého přenosu. Hlavní částí této aplikace je kontinuální snímání obrazu z připojené webkamery. Tohoto jsme dosáhli použitím funkcí

knihovny OpenCV. Na takto získané snímky jsme následně aplikovali neuronové sítě, za účelem detekce a identifikace obličejů osob na snímku. Neuronové sítě, které jsme aplikovali, jsou předtrénované neuronové sítě, které jsou součástí knihovny facenet-pytorch. Jedná se o sítě MTCNN, zajišťující detekci obličejů, a síť facenet, sloužící k identifikaci. Jelikož předtrénované neuronové sítě jsou schopny rozpoznat pouze osoby, na které byly naučeny, bylo třeba natrénovat síť k identifikaci obličejů na náš požadovaný dataset. Pro samotné přetrénování jsme použili skript, který vytvořil autor knihovny. V tomto skriptu jsme nastavili parametry trénování sítě. Jmenovitě se jednalo o velikost batche pro trénink, počet iterací pro nalezení optimálního modelu a samozřejmě cestu ke složce obsahující náš dataset (tedy fotografie). Takto natrénovanou síť jsme následně uložili do pytorch souboru, aby bylo možné jednu síť přenášet mezi různými platformami. Síť MTCNN jsme na vstupní snímek aplikovali dvakrát. Jednou za účelem získání tenzoru, obsahujícího všechny obličeje na snímku v ořezaném a upraveném tvaru. Přes tento tenzor následně iterujeme. V každé iteraci nastavíme obličej v tenzoru na aktuální pozici jako vstup sítě facenet a získáme tzv. *logits*, které nám, po převedení a nalezení maximální hodnoty, určí identitu osoby. Podruhé síť aplikujeme v jiné konfiguraci pro získání souřadnic boxů ohraničující tento obličej. Následně jsme v aplikaci vytvořili funkci, která zapisuje, kdy byla daná osoba detekována. Kontroluje také, kdy byla tato osoba naposledy spatřena. Vytvořili jsme také funkci, sloužící k vizualizaci výsledků detekce. Tato funkce vykresluje, kde se obličej nachází a do snímku také vypisuje informace o osobě. Tyto informace jsou uloženy v textových souborech, jedná se o jméno, id, věk a například zaměstnání. Vypisuje se zde také čas poslední detekce této osoby, který nám poskytuje předešlá funkce. V poslední části pak tato aplikace na monitor vykreslí snímek, včetně všech do něj vykreslených objektů.

Další směry řešení této práce by se mohly zaměřit na poslední úlohu s identifikací obličejů. Během jejího zprovoznění se objevilo několik možností dalšího řešení. Jedním směrem by mohlo být například implementování SQL databáze obsahující informace o subjektech místo textových souborů. Dalším možným směrem řešení by byla tvorba vlastní neuronové sítě, nebo například získávání informací ze snímků. Tím je myšleno například s kým se daná osoba objevila na snímku.

Použitá literatura

(1) General Python FAQ. *Python 3.8.1 documentation* [online]. Python Software Foundation, 2001 [cit. 2020-01-13]. Dostupné z: https://docs.python.org/3/faq/general.html#what-is-python

(2) Installing Packages. Https://packaging.python.org [online]. 2013 [cit. 2020-01-13]. Dostupné z: https://packaging.python.org/tutorials/installing-packages/

(3) Face Analysis. OpenCV [online]. [cit. 2020-01-13]. Dostupné z: https://docs.opencv.org/4.1.2/db/d7c/group__face.html

(4) Raspberry Pi 3 Model B. Raspberry Pi [online]. [cit. 2020-01-13]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

(5) Raspberry Pi 4 Model B – specifications. Raspberry Pi [online]. [cit. 2020-05-13]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

(6) Installing operating system images. Raspberry Pi [online]. [cit. 2020-01-13]. Dostupné z: https://www.raspberrypi.org/documentation/installation/installing-images/README.md

(7) OpenCV: About. OpenCV [online]. OpenCV team [cit. 2020-01-13]. Dostupné z: https://opencv.org/about/

(8) Image file reading and writing. OpenCV [online]. OpenCV team [cit. 2020-01-13]. Dostupné z: https://docs.opencv.org/master/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56

(9) Drawing Functions. OpenCV [online]. OpenCV team [cit. 2020-01-13]. Dostupné z: https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html

(10) High-level GUI. OpenCV [online]. OpenCV team [cit. 2020-01-13]. Dostupné z: https://docs.opencv.org/master/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563

(11) PATTERSON, Josh a Adam GIBSON. Deep learning: a practitioner's approach. Sebastopol, CA: O'Reilly, 2017. ISBN 978-1-491-91425-0.

(12) Deep Neural Network module. OpenCV [online]. OpenCV team [cit. 2020-01-13]. Dostupné z: https://docs.opencv.org/master/d6/d0f/group__dnn.html#ga29f34df9376379a603acd8df581ac8d7

(13) Samples/dnn/object_detection.py. Github [online]. OpenCV team [cit. 2020-01-13]. Dostupné z: https://github.com/opencv/opencv/blob/c722625f280258f5c865002899bf0dc2ebff1b2b/samples/dnn/object_detection.py

(14) LIN, Tsung-Yi, Michael MAIRE, Serge BELONGIE, et al. Microsoft COCO: Common Objects in Context. ArXiv. 2014.

(15) REDMON, Joseph a Ali FARHADI. YOLOv3: An Incremental Improvement. ArXiv. 2018.

(16) Cv::VideoCapture Class Reference. OpenCV [online]. OpenCV team [cit. 2020-05-01]. Dostupné z: https://docs.opencv.org/master/d8/dfe/classcv_1_1VideoCapture.html#a473055e77dd7faa4d26d686226b292c1

(17) ESLER, Tim. Facenet-pytorch. Github [online]. [cit. 2020-05-01]. Dostupné z: https://github.com/timesler/facenet-pytorch

(18) Logit. DeepAI: The front page of A.I. [online]. [cit. 2020-05-01]. Dostupné z: https://deepai.org/machine-learning-glossary-and-terms/logit

(19) ESLER, Tim. Facenet-pytorch, finetune. Github [online]. [cit. 2020-05-01]. Dostupné z: https://github.com/timesler/facenet-pytorch/blob/master/examples/finetune.ipynb

(20) TORCH CONTRIBUTORS. Torchvision.datasets. PyTorch [online]. [cit. 2020-05-01]. Dostupné z: https://pytorch.org/docs/stable/torchvision/datasets.html#torchvision.datasets.ImageFolder

(21) SHARMA, Sagar. Epoch vs Batch Size vs Iterations. Toward data science [online]. [cit. 2020-05-01]. Dostupné z: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

(22) PIL.Image.fromarray. Kite [online]. [cit. 2020-05-01]. Dostupné z: https://kite.com/python/docs/PIL.Image.fromarray

(23) Datetime. Python library [online]. Python Software Foundation [cit. 2020-05-01]. Dostupné z: https://docs.python.org/3/library/datetime.html#datetime

(24) Time. Python library [online]. Python Software Foundation [cit. 2020-05-01]. Dostupné z: https://docs.python.org/3/library/time.html#module-time

Seznam příloh